Getting Started

https://docs.fatzebra.com/docs/

Getting Started

Fat Zebra provides a simple to use REST API. It is designed to be simple, have predictable URLs and uses HTTP response codes to indicate errors. In addition to this, it uses HTTP features such as Basic Authentication and HTTP verbs (GET, PUT, POST etc). This means that it is highly compatible with most HTTP clients including `Net::HTTP`, `cURL` and `System.Net.WebClient` and `httplib`.

The API expects to receive and will respond with JSON payloads. It is recommended if you are having problems to test the formatting of your data against a lint tool, such as JSONLint (https://jsonlint.com/).

> **Note:**
>
> It is important that you ensure no sensitive data is included in the data you are validating - this includes card numbers and CVV/CVC numbers. We also recommend that you replace card holder names with example data, however names with diacritics, umlauts or accents may cause validation issues, so remember this while you are testing.

Updated over 4 years ago

Authentication →

Did this page help you? Yes No

Getting Started

https://docs.fatzebra.com/docs/getting-started#content

Getting Started

Fat Zebra provides a simple to use REST API. It is designed to be simple, have predictable URLs and uses HTTP response codes to indicate errors. In addition to this, it uses HTTP features such as Basic Authentication and HTTP verbs (GET, PUT, POST etc). This means that it is highly compatible with most HTTP clients including `Net::HTTP`, `cURL` and `System.Net.WebClient` and `httplib`.

The API expects to receive and will respond with JSON payloads. It is recommended if you are having problems to test the formatting of your data against a lint tool, such as JSONLint (https://jsonlint.com/).

> **Note:**
>
> It is important that you ensure no sensitive data is included in the data you are validating - this includes card numbers and CVV/CVC numbers. We also recommend that you replace card holder names with example data, however names with diacritics, umlauts or accents may cause validation issues, so remember this while you are testing.

Updated over 4 years ago

Authentication →

Did this page help you? Yes No

Fat Zebra Documentation

https://docs.fatzebra.com/

Home

Fat Zebra Documentation

Welcome to the Fat Zebra developer hub. You'll find comprehensive guides and documentation to help you start working with Fat Zebra as quickly as possible.

Get Started API Reference

Accepting Credit Card Payments

Credit card payments can be processed directly with the Cloud Payments API, or you can use our Direct Post or Hosted Payments option to outsource PCI DSS Compliance.

Read more about Accepting Credit Card Payments to get started.

Debit a Bank Account

Fat Zebra supports the option to direct debit a customer's bank account.

Our Direct Entry Overview details how to support this payment method.

Alternative Payment Methods

Fat Zebra supports Digital Wallets and other Alternative Payment Methods to streamline your checkout and improve your customers' security and experience.

Read about Digital Wallets.

Documentation

Documentation
Testing
Purchases
Card On File
View All...

Changelog

Improvements & Bug Fixes – August 2024
card_token parameter added for Payment Plans endpoints
Additional Authorization Reasons for Merchant Initiated Transactions
Introduction of Merchant Advice Retry After
View All...

Getting Started

https://docs.fatzebra.com/docs

Getting Started

Fat Zebra provides a simple to use REST API. It is designed to be simple, have predictable URLs and uses HTTP response codes to indicate errors. In addition to this, it uses HTTP features such as Basic Authentication and HTTP verbs (GET, PUT, POST etc). This means that it is highly compatible with most HTTP clients including `Net::HTTP`, `cURL` and `System.Net.WebClient` and `httplib`.

The API expects to receive and will respond with JSON payloads. It is recommended if you are having problems to test the formatting of your data against a lint tool, such as JSONLint (https://jsonlint.com/).

> **Note:**
>
> It is important that you ensure no sensitive data is included in the data you are validating - this includes card numbers and CVV/CVC numbers. We also recommend that you replace card holder names with example data, however names with diacritics, umlauts or accents may cause validation issues, so remember this while you are testing.

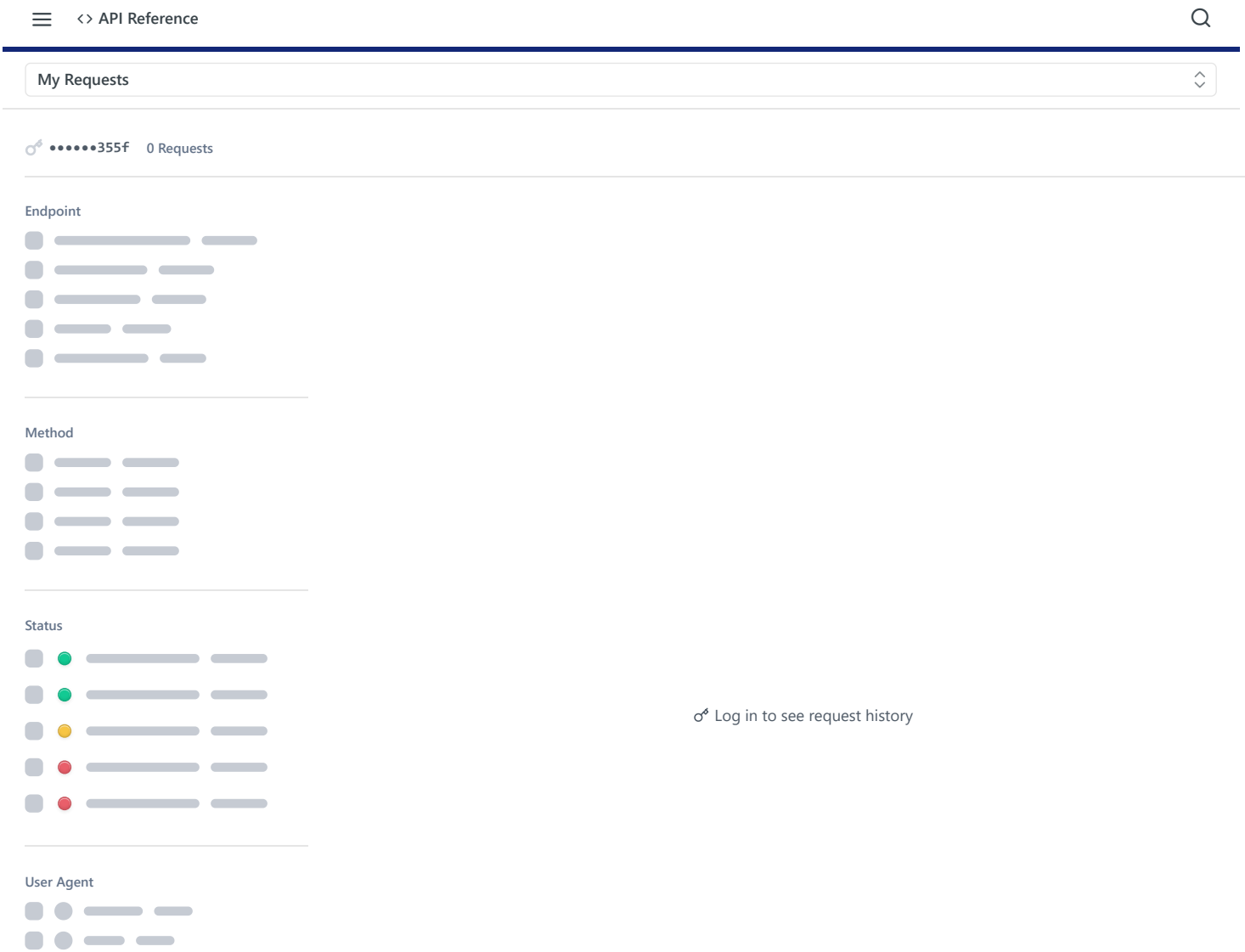
Updated over 4 years ago

Authentication →

Did this page help you? Yes No

My Requests

https://docs.fatzebra.com/reference



Changelog

https://docs.fatzebra.com/changelog

Improved

Improvements & Bug Fixes – August 2024

2 months ago by Fat Zebra Developers

Improvements

Improved

card_token parameter added for Payment Plans endpoints

9 months ago by Fat Zebra Developers

Previously, when creating a payment plan, that payment plan would use the customer's most recently updated credit card for payments.

Additional Authorization Reasons for Merchant Initiated Transactions

almost 2 years ago by Fat Zebra Developers

New authorization reasons have been added for merchant initiated transactions:

Introduction of Merchant Advice Retry After

almost 2 years ago by Fat Zebra Developers

Transactions declined with a Merchant Advice Code of `retry_later` can now refer to the `merchant_advice_retry_after` field to determine when the retry should be attempted.

Introduction of Merchant Advice Codes

almost 3 years ago by Matthew Lewis

Schemes have introduced Merchant Advice Response Codes to provide feedback to merchants on how to retry failed transactions.

fatzebra.js update 2021-05-11

over 3 years ago by Fat Zebra Developers

- Customer country field used to accept country name, e.g. Australia. It now expects to receive ISO alpha2 country code . The support for accepting country name will be dropped on 1st Nov.
- *fz.payment.success* & *fz.payment.error* events both return transaction data in full. See here for details.
- *verification* is provided in the transaction data payload for data integrity check.

New Payment Aggregator Fields

about 4 years ago by Fat Zebra Developers

Fat Zebra has added support for new Payment Aggregator fields. These fields only apply to certain merchants who have been enabled as a Payment Aggregator with their acquirer and wish to send transactions on behalf of a merchant. No change is required for all other merchants.

New Documentation Site

Welcome to the new Cloud Payments Documentation.

Sign In - Fat Zebra

https://docs.fatzebra.com/login?redirect_uri=/docs/getting-started

Sign in to your dashboard

Email

PasswordForgotten password?

Continue

Getting Started

https://docs.fatzebra.com/docs/getting-started

Getting Started

Fat Zebra provides a simple to use REST API. It is designed to be simple, have predictable URLs and uses HTTP response codes to indicate errors. In addition to this, it uses HTTP features such as Basic Authentication and HTTP verbs (GET, PUT, POST etc). This means that it is highly compatible with most HTTP clients including `Net::HTTP`, `cURL` and `System.Net.WebClient` and `httplib`.

The API expects to receive and will respond with JSON payloads. It is recommended if you are having problems to test the formatting of your data against a lint tool, such as JSONLint (https://jsonlint.com/).

> **Note:**
>
> It is important that you ensure no sensitive data is included in the data you are validating - this includes card numbers and CVV/CVC numbers. We also recommend that you replace card holder names with example data, however names with diacritics, umlauts or accents may cause validation issues, so remember this while you are testing.

Updated over 4 years ago

Authentication →

Did this page help you? Yes No

Authentication

https://docs.fatzebra.com/docs/authentication

Authentication

Authentication with the Fat Zebra API is via HTTP Basic Authentication. When your account is setup you are provided with two sets of credentials - one for the test environment (also known as the Sandbox), and one for the live system. Your test username will always be prefixed with `TEST`.

API requests must be made over HTTPS - any requests over HTTP will fail. All requests require authentication.

> **Note:**
>
> It is important to note that your username is not case sensitive however your API token (password) is.

cURL

```
$ curl https://gateway.pmnts-sandbox.io/v1.0/purchases -u TEST:TEST
```

Updated over 4 years ago

← Getting Started

Errors & Timeouts →

Did this page help you? Yes No

Errors & Timeouts

https://docs.fatzebra.com/docs/error-codes

Errors & Timeouts

Fat Zebra uses standard HTTP response codes to indicate success or failure of an API request. In general, codes in the 2XX range indicate success. Codes in the 4XX range indicate an error based on the information provided (e.g., a required parameter was omitted, a charge failed). Codes in the 5XX range indicate an error with Fat Zebra's systems.

Some 4XX errors can be handled programmatically (e.g., a card is declined) and include an error code that briefly explains the error reported.

HTTP status code summary

Status Code	Summary
200 - OK	The request was successfully completed.
201 - Created	The request was successful and a transaction was created.
400 - Bad Request	Bad data was received.
401 - Unauthorised	Your API credentials were not valid.
403 - Forbidden	The resource you were requesting is not available for your credentials. You will most commonly see this is you are attempting to fetch a payment or refund created by your live credentials with your test credentials.
404 - Not Found	The requested object was not found.
422 - Unprocessable Entity	The data provided was unprocessable, however the syntax of the request was correct.
500, 502, 503, 504 - Server Errors	Something went wrong with Fat Zebra's systems or the underlying infrastructure.

Timeouts

When making requests to the Gateway it is important to understand transaction processing timeouts to ensure the best customer experience possible.

Fat Zebra sets a strict timeout of 45 seconds on transactions being processed against the card networks - this means that from the time a request arrives in Fat Zebra's systems, it is permitted to process for up to 45 seconds before the Gateway stops the transaction, and a timeout response sent back to the merchant systems. It is suggested that merchants allow up to 50 seconds for a transaction to complete before retrying.

When a transaction is timed out by the Gateway our systems will automatically issue a transaction reversal, to void any in-flight transactions in the event it may have been successfully processed.

If a merchant chooses to use a lower timeout when communicating with the Gateway it is important to ensure that a query and subsequent void is issued against the transaction reference, to avoid any orphaned transactions impacting card holders.

Endpoint Base URLs

https://docs.fatzebra.com/docs/endpoint-base-urls

Endpoint Base URLs

There are two base URLs for interacting with the Gateway, depending on the environment required:

Environment	URL
Sandbox (Test)	https://gateway.pmnts-sandbox.io/
Production (Live)	https://gateway.pmnts.io/

Where merchants require statically addressed endpoints (such as for IP Whitelisting of outbound network traffic to the internet) the following endpoints can be used, however merchants must understand that due to the nature of these endpoint configurations ***there may be a longer delay during a network failover in the event of a technical fault, compared to the above endpoints.*** This is usually around 60 seconds as opposed to 10-30 seconds normally. If this is of concern to you please contact our Support team to discuss further.

Environment	URL	IP addresses
Sandbox (Test)	https://gateway-static.pmnts-sandbox.io	75.2.69.3 99.83.178.72
Production (Live)	https://gateway-static.pmnts.io	75.2.13.242 99.83.244.137

For the Hosted Payment Page the following URLs apply:

Environment	URL
Sandbox (Test)	https://paynow.pmnts-sandbox.io/
Production (Live)	https://paynow.pmnts.io/

Updated over 4 years ago

← Errors & Timeouts

Pagination →

Did this page help you? Yes No

Pagination

https://docs.fatzebra.com/docs/pagination

Pagination

In order to keep responses quick and small, multi-value results will be paginated.

By default a request will return up to 10 records per page, however this can be specified to return up to 50 per request.

To page through the records, each request will need to provide the following values in the query string:

Name	Type	Description
`limit`	integer	The number of records to return per request. Defaults to 10, maximum 50.
`offset`	integer	The offset (i.e. page number) to return. Defaults to 1.

For example, if your limit is set to 10, and you wish to retrieve records 101 to 110 you would specify the following parameters in your request:

```
limit=10&offset=11
```

The API will return a maximum of 50 records even if a limit greater than 50 is requested.

Updated over 2 years ago

← Endpoint Base URLs

Glossary →

Did this page help you? Yes No

Glossary

https://docs.fatzebra.com/docs/glossary

Glossary

Ecommerce (and banking.. and life) is full of terms which you might not understand - we've certainly expanded our vocabulary in building an Internet Payment Gateway.

To make things easier here is a collection of some of the terms you might come across dealing with Internet Payments, Internet Merchant Facilities and Credit Card payments in general.

Acceptor
Any party which accepts credit card transactions.

Access Code
Commonly used by MiGS - this is part of the authentication for your merchant account.

Acquirer
Also referred to as "acquiring bank" or "acquiring financial institution". An acquirer is an entity that initiates and maintains relationships with merchants for the acceptance of payment cards.

Authentication
Verification that someone is who they say they are. When you log in to a website with a username and password the process behind that is called Authentication.

Authorization
Approval of a transaction by or on behalf of an issuer according to defined operations regulations. The merchant receives, via telephone or authorisation terminal, this approval to process the transaction.

CVV/CV2/CSV/CVC/CAV/CSC/Security Code
Also known as Card Validation Code or Value, or Card Security Code. This commonly refers to the 3 or 4 digit code printed on the back of the card on the signature panel (or on the front of the card for an American Express card).

Capture
The process of completing a pre-authorised transaction.

Chargeback
A dispute resolution process that members use to determine the responsible party in a chargeback related dispute.

Clearing
The facilitation of funds transfer between institutions.

Cryptography
Discipline of mathematics and computer science concerned with information security, particularly encryption and authentication. In applications and network security, it is a tool for access control, information confidentiality, and integrity. In short, principals of cryptography is used to make sensitive data unreadable.

e-commerce
Electronic commerce, commonly known as e-commerce refer to the buying and selling of products or services over electronic systems such as the internet and other computer networks.

EFTPOS
Electronic Funds Transfer at Point of Sale - the general term used for debit card based systems for processing transactions through terminals at points of sale.

Encryption
The process of transforming information using an algorithm to make it unreadable to anyone accept those possessing special knowledge.

Financial Institution
Any commercial bank, federal or state savings and loan association, federal or state savings bank, or credit union.

Fraud
Fraud is an intentional deception made for personal gain or to damage another individual.

HTTPS
Acronym for "hypertext transfer protocol over secure socket layer". Secure HTTP that provides authentication and encrypted communication on the World Wide Web designed for security-sensitive communication such as web-based logins.

Hashing
Process of rendering cardholder data unreadable by converting data into a fixed-length message digest via Strong Cryptography. Hashing is a (mathematical) function in which a non-secret algorithm takes any arbitrary length message as input and produces a fixed length output (usually called a "hash code" or "message digest"). A hash function should have the following properties:

- It is computationally infeasible to determine the original input given only the hash code.
- It is computationally infeasible to find two inputs that give the same hash code.

Interchange
The exchange of transaction data between acquirers and issuers.

Also refers to the fee involved between a merchant bank and a card scheme, usually passed on to the merchant (this is usually around 1-2%).

JSON
JavaScript Object Notation - a lightweight data interchange format which is easy for humans to read and write. JSON is the data format used in the Fat Zebra API.

JavaScript
A scripting language commonly used within website to provide client side programming support. Part of the foundation for JSON.

MOTO
Mail Order/Telephone Order. A transaction initiated by mail or telephone to be debited or credited to a bankcard account.

Merchant
A merchant is defined as any entity that accepts payment cards bearing the logos of any of the five members of PCI SSC (American Express, Discover, JCB, MasterCard or Visa) as payment for goods and/or services.

Merchant Bank
A bank that has entered into an agreement with a merchant to accept deposits generated by bankcard transactions; also called the acquirer or acquiring bank.

Merchant ID
A unique number assigned by the acquirer to identify the merchant.

MiGS
Mastercard Internet Gateway Service.

PAN
Acronym for "primary account number" and also referred to as "account number" or "card number". The PAN is a unique card number (typically for credit or debit cards) that identifies the issuer and the particular cardholder account.

PCI-DSS
Payment Card Industry - Data Security Standards is an open global forum launched in 2006 that is responsible to enhance Payment Card Industry data security.

The councils five founding global payment brands are American Express, Discover Financial Services, JCB International, Mastercard Worldwide and Visa Inc.

Payment Gateway
A third party which handles the interaction between a merchant and the acquiring bank in a secure environment. Payment gateways commonly provide additional flexibility and functionality then direct integration with your bank, and allow for your to change banks without changing your integration.

Recurring Payments
Payments by an issuer to an acquirer on behalf of a cardholder who authorises a merchant to bill the cardholder's account on a recurring basis (such as monthly or quarterly). The amount of each payment may be the same or may fluctuate. Also referred to as a pre-authorised order (not to be confused with pre-authorization for transactions).

Refund
Opposite of a purchase transaction; namely, the cardholder returns goods to the merchant and is credited for their value. Positive interchange and merchant service charge are reversed.

Reversal
See Chargeback.

SSL
Acronym for "Secure Sockets Layer". SSL is an established industry standard that encrypts the channel between a web browser and web server to ensure the privacy and reliability of data transmitted over this channel. Most people would know SSL as HTTPS, or recognise the 'padlock' icon in the browser indicating a website is secure.

Scheme
Card Scheme refers mainly to Visa and MasterCard, as the owners of the payment scheme, into which a bank or any other eligible financial institution can become a member. Other schemes include American Express, JCB, Discover and more.

Secure Code/Verified by Visa/3D Secure
3D Secure, which is also known MasterCard SecureCode and Verified by VISA (VbV) is a mechanism designed to provide an additional layer of security and fraud prevention for online transactions. Usually when 3D Secure is used the customer will be prompted for a password they have setup, on a web page hosted by their bank or an approved third party. Also now known as Secure Cardholder Authentication.

Settlement
The deposit of cleared funds into the merchants designated account (settlement account). Settlement usually occurs late at night, or is considered 'next day', however some merchant banks in Australia now offer real-time settlement.

Sweep Account
Also known as a settlement account. Generally funds are deposited here and transferred to an assets account.

Terminal ID
A unique number assigned by the acquirer to identify the terminal.

Token
A value which is not considered sensitive - used in lieu of sensitive data as an identifier or reference.

Tokenization
Tokenization is the process of replacing some piece of sensitive data with a value that is not considered sensitive in the context of the environment that consumes the token and the original sensitive data.

Void
The process of canceling a pre-authorised transaction (voiding).

Updated over 4 years ago

← Pagination

Test Card Numbers →

Did this page help you? Yes No

Test Card Numbers

https://docs.fatzebra.com/docs/test-card-numbers

Test Card Numbers

We provide test card numbers for use in our sandbox environment. Each of the following card numbers has a predetermined purchase response code.

> **Important: Use Test Card Numbers Only**
>
> It is important that Merchants, Developers and Testers **only** use the provided test card numbers in the Fat Zebra Sandbox. If live card numbers are used the transactions will not be processed successfully.

For example, every purchase made with VISA 4005 5500 0000 0001 will respond with 00 Approved, and every purchase made with VISA 4557 0123 4567 8902 will respond with 05 Declined.

Scheme	Card number	Purchase response code
Mastercard	5123 4567 8901 2346	00 Approved
	5313 5810 0012 3430	05 Declined
VISA	4005 5500 0000 0001	00 Approved
	4557 0123 4567 8902	05 Declined
AMEX	3456 789012 34564	00 Approved
	3714 496353 98431	05 Declined
JCB	3530 1113 3330 0000	00 Approved
	3566 0020 2036 0505	05 Declined
China UnionPay	6250946000000016	00 Approved
	6250947000000014	05 Declined

We also provide test card numbers with which you can set the purchase response code using the last two digits of the purchase amount.

For example, a purchase of AUD$50.09 with any card number below will respond with 09 Acquirer Busy.

Refer to our list of Response Codes.

Scheme	Card number	Purchase response code
Mastercard	5555 5555 5555 4444	Last two digits of purchase amount.
	2221 0012 3456 7896	
VISA	4242 4242 4242 4242	
	4111 1111 1111 1111	
	4000 0012 3456 2345 678 (19 digit card number)	
AMEX	3700 000000 00002	
	3760 701663 31008	
China UnionPay	6222 8212 3456 0017	
Discover	6011 3325 6061 8887	
	6011 1111 1111 1117	
Diners	3670 01020 00000	

Updated over 1 year ago

Glossary

Overview

Did this page help you? Yes No

Overview

https://docs.fatzebra.com/docs/purchases-overview

Overview

Credit Card payments can be processed directly with the Fat Zebra API, or you can use our Direct Post or Hosted Payments option to outsource PCI DSS Compliance. Please find links below to our integration options.

- Hosted Payment Pages
- Payments API
- Direct Post

We support real time payments as well as the option to tokenise a card, and process the transaction later. Please find links below to our integration options when processing a transaction with a card token.

- Tokenized Credit Cards API
- Batch Payments

Please find links below for further information regarding supported features for credit card payments.

- Fraud Screening
- Refunds
- Surcharging
- Recurring
- 3D Secure Card Payments
- Metadata

Updated over 2 years ago

← Test Card Numbers

Response Codes →

Did this page help you? Yes No

Response Codes

https://docs.fatzebra.com/docs/response-codes

Response Codes

Purchase, Refund and Auth/Captured responses from Fat Zebra include a `response_code` value which maps to a standard list of responses from the financial network. These codes, their descriptions and the status of the transaction are detailed below:

Code	Message/Description (Returned from Gateway)	Bank Meaning	Successful?	Next Steps
00	Approved	Approved	Yes	
01	Refer to Card Issuer	Transaction Declined	No	The customer needs to contact their card issuer for more information.
02	Refer to Card Issuer	Transaction Declined	No	The customer needs to contact their card issuer for more information.
03	No Merchant	Merchant configuration error	No	Check the merchant details provided.
04	Refer to Card Issuer	Card marked as Lost or Stolen	No	The specific reason for the decline should not be reported to the customer. Instead, it needs to be presented as a generic decline.
05	Refer to Card Issuer	Do Not Honor	No	The customer needs to contact their card issuer for more information.
06	Merchant/Acquirer Error	Error - Original Not Found or similar	No	
07	Refer to Card Issuer	Card marked as lost or stolen (Pick Up Card - Special)	No	The specific reason for the decline should not be reported to the customer. Instead, it needs to be presented as a generic decline.
08	Approved	Honour with Identification	Yes	
09	Acquirer Busy	Request in Progress (Local Terminal Error)	No	
10	Approved	Approved for Partial Amount	Yes	
11	Approved	Approved - VIP	Yes	
12	Invalid Transaction	Invalid Transaction Type	No	The customer needs to contact their card issuer to make sure their card can be used to make this type of purchase.
13	Invalid Amount	Invalid Amount	No	If the amount appears to be correct, the customer needs to check with their card

Code	Message/Description (Returned from Gateway)	Bank Meaning	Successful?	Next Steps
				issuer that they can make purchases of that amount.
14	Invalid Card Number	Invalid Card Number	No	The customer needs to contact their card issuer to check that the card is working correctly.
15	No Issuer	The card does not have a valid issuer in the scheme registration file	No	
16	Approved	Approved - Update Track 3	Yes	
17	Declined	Not Used	No	
18	Declined	Not Used	No	
19	Declined	Re-enter last transaction	No	The payment should be attempted again. If it still cannot be processed, the customer needs to contact their card issuer.
20	Declined	Not Used	No	
21	Declined	Not Used	No	
22	Declined	Suspected Malfunction	No	The payment should be attempted again. If it still cannot be processed, try again later.
23	Declined	Unacceptable Transaction Fee	No	
24	Declined	Not Used	No	
25	Declined	Unable to Locate Record on File	No	
26	Declined	Not Used	No	
27	Declined	Not Used	No	
28	Declined	Not Used	No	
29	Declined	Not Used	No	
30	Declined	Format Error	No	
31	Bank Not Supported	Bank Not Supported by Switch/Route	No	
32	Declined	Not Used	No	
33	Expired Card	Expired Card (Capture)	No	The customer should use another card.
34	Declined	Suspected Fraud - Retain Card	No	The specific reason for the decline should not be reported to the customer. Instead, it needs to be presented as a generic decline.
35	Declined	Card Acceptor, Contact Acquirer, Retain Card	No	The specific reason for the decline should not be reported to the customer. Instead, it needs to be presented as a generic decline.
36	Declined	Restricted Card - Retain Card	No	The specific reason for the decline should not be reported to the customer. Instead, it needs to be

Code	Message/Description (Returned from Gateway)	Bank Meaning	Successful?	Next Steps
				presented as a generic decline.
37	Declined	Contact Acquirer Security Department, Retain Card	No	The specific reason for the decline should not be reported to the customer. Instead, it needs to be presented as a generic decline.
38	Declined	PIN Retries Exceeded (Capture Card)	No	The customer must use another card or method of payment.
39	Declined	No Credit Account	No	The customer needs to contact their card issuer to check that the card is working correctly.
40	Declined	Function Not Supported	No	
41	Declined	Lost Card	No	The specific reason for the decline should not be reported to the customer. Instead, it needs to be presented as a generic decline.
42	Declined	No Universal (Credit-capable) account	No	The customer needs to contact their card issuer to check that the card is working correctly.
43	Declined	Stolen Card	No	The specific reason for the decline should not be reported to the customer. Instead, it needs to be presented as a generic decline.
44	Declined	No Investment Account (Not Used)	No	The customer needs to contact their card issuer to check that the card is working correctly.
45	Declined	Not Used	No	
46	Declined	Not Used	No	
47	Declined	Not Used	No	
48	Declined	Not Used	No	
49	Declined	Not Used	No	
50	Declined	Not Used	No	
51	Insufficient Funds	Insufficient Funds	No	The customer should use an alternative payment method.
52	Declined	No Cheque Account	No	The customer needs to contact their card issuer to check that the card is working correctly.
53	Declined	No Savings Account	No	The customer needs to contact their card issuer to check that the card is working correctly.
54	Expired Card	Expired Card	No	The customer should use another card.
55	Declined	Incorrect PIN	No	The customer should try again using the correct PIN.

Code	Message/Description (Returned from Gateway)	Bank Meaning	Successful?	Next Steps
56	Declined	No Card Record - Check with Issuer	No	The customer needs to contact their card issuer to check that the card is working correctly.
57	Declined	Function Not Permitted to Card Holder	No	The customer needs to contact their card issuer to make sure their card can be used to make this type of purchase.
58	Declined	Function Not Permitted to Terminal	No	
59	Declined	Suspected Fraud	No	The specific reason for the decline should not be reported to the customer. Instead, it needs to be presented as a generic decline.
60	Declined	Merchant to Contact Acquirer	No	
61	Declined	Exceeds Withdrawal Limit	No	
62	Declined	Restricted Card	No	
63	Declined	Not Used	No	
64	Declined	Not Used	No	
65	Declined	Exceeds Withdrawal	No	The customer's card issuer has declined the transaction as the customer has exceeded the withdrawal frequency limit.
66	Declined	Not Used	No	
67	Declined	Not Used	No	
68	Declined	Not Used	No	
69	Declined	Not Used	No	
70	Declined	Not Used	No	
71	Declined	Not Used	No	
72	Declined	Not Used	No	
73	Declined	Not Used	No	
74	Declined	Not Used	No	
75	Declined	PIN Tries Exceeded	No	The customer must use another card or method of payment.
76	Declined	Not Used	No	
77	Declined	Not Used	No	
78	Declined	Not Used	No	
79	Declined	Not Used	No	
80	Declined	Not Used	No	
81	Declined	Not Used	No	
82	Declined	CVV Validation Error	No	The customer should try again using the correct CVV.
83	Declined	Not Used	No	
84	Declined	Not Used	No	
85	Declined	Not Used	No	

Code	Message/Description (Returned from Gateway)	Bank Meaning	Successful?	Next Steps
86	Declined	Not Used	No	
87	Declined	Not Used	No	
88	Declined	Not Used	No	
89	Declined	Not Used	No	
90	Declined - Please Retry	Cutoff In Progress - retry allowed	No	The payment should be attempted again. If it still cannot be processed, the customer needs to contact their card issuer.
91	Declined	Host or Switch Unavailable/Card Issuer Signed Off/Card Issuer Timed Out/Card Issuer Unavailable	No	The payment should be attempted again. If it still cannot be processed, the customer needs to contact their card issuer.
92	Declined	Unable to Route Transaction	No	The payment should be attempted again. If it still cannot be processed, try again later.
93	Declined	Cannot Complete, Violation Of The Law	No	The specific reason for the decline should not be reported to the customer. Instead, it needs to be presented as a generic decline.
94	Declined	Duplicate Transaction	No	Check to see if a recent payment already exists.
95	Declined	Not Used	No	
96	Declined	System Error	No	The payment should be attempted again. If it still cannot be processed, try again later.
99	System Error - Contact Gateway if error persists	System Error	No	The payment should be attempted again. If it still cannot be processed, try again later.

Updated over 1 year ago

← Overview

Metadata →

Did this page help you? Yes No

Metadata

https://docs.fatzebra.com/docs/metadata

Metadata

When creating purchases, it's possible to store additional keys and values against each purchase. These values will be returned when querying the purchase in future, and are also visible via the Merchant Dashboard. Attaching metadata is done using the `metadata` key, as demonstrated below.

Metadata Example

```
{
  "metadata": {
    "your_reference": "abcd1234",
    "something_else": 42
  }
}
```

Updated almost 6 years ago

Response Codes

Addendum Data

Did this page help you? Yes No

Addendum Data

https://docs.fatzebra.com/docs/addendum-data

Addendum Data

For some switching paths and acquirers it may be possible to include addendum data which is passed onto the card issuer - this is primarily used for corporate purchasing cards, American Express cards etc.

To include this data with your transactions, the field `addendum_data` should be added to the request payload with the appropriate data payload for the transaction type - these different payload types are detailed below.

JSON

```
{
  "card_holder": "Jim Smith",
  "card_number": "5123456789012346",
  "card_expiry": "05/2023",
  "cvv": "987",
  "amount": 1000,
  "reference": "ORD98976",
  "customer_ip": "111.222.111.123",
  "currency": "AUD",
  "addendum_data": {
    "corp_l2": {
      "cardmember_reference": "ORD 123456 PO 08789",
      "description_1": "Box of paper",
      "quantity_1": 5,
      "total_1": 5000
    }
  }
}
```

Addendum data is accepted for Purchase (Pre-auth and Purchase), and Captures, and should be included in the normal request payload, for example to include addendum data in a capture request:

JSON

```
{
  "amount": 10000,
  "addendum_data": {
    "corp_l2": {
      "cardmember_reference": "ORD 123456 PO 08789",
      "description_1": "Box of paper",
      "quantity_1": 5,
      "total_1": 5000
    }
  }
}
```

If addendum data was presented with the original transaction (such as for a pre-auth) the data will be merged, with the most recent data overwriting the older data.

Corporate Purchasing Level 2 Data

Level 2 data for corporate purchase cards may be included which can detail the order reference, purchase order numbers, line item details etc.

Field Name	Type (Length)	Description
cardmember_reference	String (20), Alphanumeric and Spaces only	The card member's reference number, such as a store order number and/or customer PO Number

Field Name	Type (Length)	Description
description_1 - description_4	String (40), Alphanumeric and Spaces only	The description for (up to) the first four line items. These should be descriptive and avoid generic terms such as 'Merchandise'
quantity_1 - quantity_4	Integer (3)	The quantity of items for each line item
shipping_postcode	String (15), Alphanumeric and Spaces only	The post code for the shipping address
total_1 - total_4	Integer (5)	The total for each line item as a whole number in the smallest unit (e.g. $155.60 will be 15560)

Full Corporate Level 2 data example:


```
{
  "corp_l2": {
    "cardmember_reference": "ORD 123456 PO 08789",
    "description_1": "Box of paper",
    "quantity_1": 5,
    "total_1": 5000,
    "description_2": "Pencils",
    "quantity_2": 10,
    "total_2": 1000,
    "description_3": "Pens",
    "quantity_3": 10,
    "total_3": 1500,
    "description_4": "White Out",
    "quantity_4": 1,
    "total_4": 500,
    "shipping_postcode": "EC1A 1AA"
  }
}
```


Updated about 6 years ago

← Metadata

Extra and Extended Fields →

Did this page help you? Yes No

Extra and Extended Fields

https://docs.fatzebra.com/docs/extra-and-extended-fields

Extra and Extended Fields

Fat Zebra enables extended fields for merchants to allow advanced integrations where the need arises. These fields include 3D Secure values (XID, CAVV, PARes, VERes, SLI), ECM and Dynamic Descriptors. If you require this functionality for your integration please contact Fat Zebra support and detail your requirements so we can determine the best option for your needs.

All relevant fields will only be accepted for merchants which are explicitly configured to support extra fields.

All fields should be included under the extra key in the request payload:

Example

```
{
    "extra": {
        "sli": "09"
    }
}
```

Updated almost 6 years ago

Addendum Data

Card On File

Did this page help you? Yes No

Card On File

https://docs.fatzebra.com/docs/recurring-and-other-transaction-types

Card On File

Card on file transactions include:

1. Tokenizing credit card for use later. (See example here)
2. Recurring and instalment payment. (See example here)
3. Merchant initiated transactions. (See example here)

There are several fields related to these transactions.

ECM

The ECM (eCommerce Indicator) describes the card transaction type. The field is two digits (mn) with the possible values being:

First Digit (m)	Second Digit (n)
Telephone Order - 1	Single - 1
	Recurring - 2
	Instalment - 3
Mail Order - 2	Single - 1
	Recurring - 2
	Instalment - 3
Internet Order - 3	Single - 1
	Recurring - 2
	Instalment - 3

Below is an example on how to set the ECM value.

Example: Recurring Payment

```
{
  "extra": {
    "ecm": 32
  }
}
```

Stored Credential Indicator

The stored credential indicator is used to indicate if this is the initial recurring / instalment transaction, or the subsequent transaction.

Description	Value
Initial Transaction	I
Subsequent Transaction	S

Below is an example on how to use stored credential indicator

JSON

```
{
  "extra": {
    "stored_credential_indicator": "I"
```

```
  }
}
```

Omit Expiry

The omit expiry field set to true can also be sent through for recurring transactions against expired cards. This will allow the transaction to be sent to the banking network without the expiry date field. Typically, this field increases approval rates for recurring transactions against expired cards.

Example: Omit Expiry Flag

```
{
  "extra": {
    "omit_expiry": true,
    "ecm": 32
  }
}
```

Merchant Initiated Transactions

A merchant initiated transaction is a payment that is agreed with a payer's consent, and is initiated by the merchant collecting the payment.

In order to apply for a merchant initiated transaction, the merchant need to specify why this transaction is initiated using the `auth_reason` field. Below are available reasons.

Authorization Reason	Value
Resubmission Payment	resubmission
Delayed Charges Payment	delayed_charges
Re-authorization Payment	reauthorization
No Show	no_show
Account Topup	account_topup
Unscheduled Payment	unscheduled
Incremental	incremental
Instalment	instalment
Recurring Transaction - Subscriptions*	subscription
Partial Shipment	partial shipment

[*] Subscriptions are a recurring payment of fixed value

Example: No Show

```
{
  "extra": {
    "auth_reason": "no_show"
  }
}
```

Card On File

The card on file flag set to true indicates we are using a credit card that is stored in a vault, eg. card that is tokenized via Fat Zebra gateway. All recurring, instalment payment and merchant initiated transactions are card on file transactions.

Note that you don't need to apply this value for recurring, installment payment and merchant initiated transaction, as the system will apply it automatically.

Example: Card On File

```
{
  "extra": {
    "card_on_file": true
  }
}
```

Updated almost 2 years ago

Extra and Extended Fields

3D Secure Card Payments

Did this page help you? Yes No

3D Secure Card Payments

https://docs.fatzebra.com/docs/3d-secure

3D Secure Card Payments

3D Secure card payment data can be added to authorizations, purchases, and refunds. All fields should be included under the extra key in the request payload.

The values for `sli`, `xid`, `cavv`, `par` and `ver` should be supplied by the 3D Secure MPI being used for transaction authentication.

If you do not have an MPI, Fat Zebra can provide an MPI. Refer to the 3DS2 Integration documentation for more information.

Field name	Type	Description
`sli`	String (2-digits)	The Security Level Indicator (SLI) indicates the type of card holder authentication used for the transaction. Accepted values are described in the table bellow.
`xid`	String	The XID is the transaction ID from the 3D Secure provider. This is Base64 encoded data.
`cavv`	String	The CAVV is the card authentication verification value, provided by the 3D Secure Provider. This is Base64 encoded data.
`par`	String	This is the `PARes` value from 3D Secure
`ver`	String	This is the `VERes` value from 3D Secure
`threeds_version`	String	3DS version used for authenticating the card. The version may include the major, minor, and build numbers, e.g. `2`, `2.1`, or `2.1.0` are all accepted values. If omitted, this field will default to a value of `1`.
`directory_server_txn_id`	String	Directory Server Transaction ID from the 3D Secure provider. This is commonly a string in UUID format.

Visa & eftpos SLI	Mastercard SLI (MPI compatibility)	Description
`"05"`	`"02"`	Secure, Authenticated transaction with XID and CAVV present
`"06"`	`"01"`	Secure, Non-authenticated transaction (XID present, CAVV not present)
`"07"`	`"00"`	Secure, Non-authenticated transaction (XID and CAVV not present). This is the default value.

Note: All fields should be included under the extra key in the request payload.

Example: 3DS payment data

```
{
    "extra": {
        "sli": "05",
        "cavv": "MzM2OGI2ZjkwYjYwY2FjODQ3ZWU=",
        "xid": "ZGUzNzgwYzQxM2ZlMWM0MzVkMjc=",
        "par": "Y",
        "ver": "Y",
        "directory_server_txn_id": "5ddb4c13-2e30-4901-9854-5f0305097a25",
        "threeds_version": "2.1.0"
    }
}
```

Updated 3 months ago

← Card On File

Mail Order / Telephone Order Payments →

Did this page help you? Yes No

Mail Order / Telephone Order Payments

https://docs.fatzebra.com/docs/mail-order-telephone-order-payments

Mail Order / Telephone Order Payments

Mail Order / Telephone Order Payments should be identified using the `ecm` indicator. The field is two digits (mn) with the possible values being:

First Digit (m)	Second Digit (n)
Telephone Order - 1	Single - 1
	Recurring - 2
	Instalment - 3
Mail Order - 2	Single - 1
	Recurring - 2
	Instalment - 3
Internet Order - 3	Single - 1
	Recurring - 2
	Instalment - 3

Telephone Order

```
{
  "extra": {
    "ecm": 11
  }
}
```

JSON

```
{
  "extra": {
    "ecm": 21
  }
}
```

Updated over 2 years ago

← 3D Secure Card Payments

Dynamic Descriptors →

Did this page help you? Yes No

Dynamic Descriptors

https://docs.fatzebra.com/docs/dynamic-descriptors

Dynamic Descriptors

Specifies the card acceptor descriptor details to show up on the cardholder's statement. This is currently only supported for a handful of banks, and must be enabled before use.

Dynamic Descriptors are specified by the field **descriptor** of type **Hash**.
Required "descriptor" fields are **name** and **location** and they are both of type **String**. Alternatively you can provide the **raw** descriptor field in lieu of **name** and **location**.

Note: If the length of the values exceeds the maximum you will receive a validation error. It is also important to note that the values may be truncated or reflected differently based on how card issuers display these values to their customers.

Field	Type	Description
name	String	The card descriptor name to show on the customers statement. Maximum 20 characters
location	String	The card descriptor location to show on the customers statement. Maximum 13 characters
raw	String	A raw descriptor value which can be presented instead of the name and location values. Maximum 35 characters.

Example: Dynamic Descriptors | Example: Raw Descriptor

```
{
  "extra": {
    "descriptor": {
      "name": "Biz Name Pty",
      "location": "Surry Hills"
    }
  }
}
```

Notes for Bill Payment Service Providers/Consumer Bill Payment Services

Merchants who process transactions for bill payments on behalf of other companies, known as Bill Payment Service Providers or Consumer Bill Payment Services, are required to format their descriptor according to scheme requirements, as follows:

`BPSP NAME*MERCHANT NAME`

In order to do this the `raw` descriptor field must be used, as per the following example:

Text

```
{
  "extra": {
    "descriptor": {
      "raw": "PAYCORP*Clean Co"
    }
  }
}
```

Updated about 2 years ago

← Mail Order / Telephone Order Payments

Payment Aggregators →

Did this page help you? Yes No

Payment Aggregators

https://docs.fatzebra.com/docs/payment-aggregators

Payment Aggregators

> **Limited Acquirer Support For Payment Aggregators**
>
> Processing transactions as a Payment Aggregator is not supported by all Australian acquirers. If these methods are used where support is not yet available the transactions may be declined or return errors. Contact the Fat Zebra support team for more information.

The Payment Aggregator fields allow merchants who act as a Payment Aggregator to specify sub-merchant details required by their acquirers. This is currently only supported for a handful of banks, and must be enabled before use by contacting the Fat Zebra support team.

Payment Aggregator fields are specified in the field **sub_merchant** of type **Hash**. They can be specified on authorisation, purchase, and refund transactions.

Field	Type	Description
id	String *(max 20 characters)*	The sub merchant's id as specified by your acquirer This field is mandatory for Westpac merchants
aggregator_name	String *(max 38 characters)* *Note: **aggregator_name** + **name** must not exceed 37 characters*	The aggregator name assigned to you by your acquirer For Westpac merchants this is a 3 character string
name	String *(max 38 characters)* *Note: **aggregator_name** + **name** must not exceed 37 characters*	The business name of the sub merchant
address	String *(max 38 characters)*	The street address of the sub merchant This field is mandatory for Westpac merchants
city	String *(max 21 characters)*	The city of the sub merchant
state	String *(max 3 characters)*	The state of the sub merchant
postcode	String *(max 15 characters)*	The postcode of the sub merchant This field is mandatory for Westpac merchants
country_code	String *(max 2 characters)*	The country code of the sub merchant in 2-letter ISO 3166-2 E.g. use **AU** for Australia
phone	String *(max 20 digits)*	The merchants phone number (for customers to contact if required)
email	String *(max 40 characters)	The merchants email address (for customers to contact if required)

Field	Type	Description
		Optional

Example: Payment Aggregator fields

```
{
  "extra": {
    "sub_merchant": {
      "id": "ABC",
      "aggregator_name": "FZ",
      "name": "ABC Pty Ltd",
      "address": "58 Kippax Street",
      "city": "Surry Hills",
      "state": "NSW",
      "postcode": "2010",
      "country_code": "AU",
      "email": "contact@abc.com.au",
      "phone": "0290371840"
    }
  }
}
```

Updated over 1 year ago

← Dynamic Descriptors

Remittance Merchants →

Did this page help you? Yes No

Remittance Merchants

https://docs.fatzebra.com/docs/remittance-overview

Remittance Merchants

Remittance Parameters

Money remittance merchants (Merchant Category Code `4829` or `6540`) are required to provide additional data in the request payload under `extra` for these transactions:

- Purchase
- Auth
- Refund (standalone)

JSON

```
{
  "extra": {
    "remittance": {
      "bai": "p2p",
      "sender": {
        "first_name": "Rachel",
        "last_name": "Remitter",
        "address": "1 Remittance Road",
        "city": "Sydney",
        "country": "AUS"
      },
      "recipient": {
        "first_name": "Richard",
        "last_name": "Remittee",
        "country": "NZL",
        "account_type": "06",
        "account_number": "6666667777777333"
      }
    }
  }
}
```

Business Application Identifier (BAI)

Visa

Sender	Receiver	Merchant supports Visa-OCT?	BAI
Person	Person (different)	YES	`p2p`
Person	Person (different)	NO	`p2p_card`
Person	Person(same)		`self`

Mastercard

Sender	Receiver	Receiver Account Type	BAI
Person	Person (different)	NOT CARD	`p2p`
Person	Person (different)	IS CARD	`p2p_card`
Person	Person (same)	NOT CARD	`self`
Person	Person (same)	IS CARD (debit/prepaid)	`self_debit_card`
Business	Business (same)		`b2b`
Business	Business (different)		`b2b_card`

Sender Data

Field	Type	Length
`first_name`	String	Max: 35
`last_name`	String	Max: 35
`address`	String	Max: 35
`city`	String	Max: 25
`country`	String	3 (ISO 3166-1 alpha-3)

Recipient Data

Account Type

Field	Type	Length
`first_name`	String	Max: 35
`last_name`	String	Max: 35
`country`	String	3 (ISO 3166-1 alpha-3)
`account_type`	String	2 `00` – Other `01` – RTN + Bank Account `02` – IBAN `03` – Card Account `04` – Email `05` – Phone Number `06` – Bank Account Number (BAN) + Bank Identification Code (BIC) `07` – Wallet ID `08` – Social Network ID
`account_number`	String	Max: 50

Updated over 1 year ago

← Payment Aggregators

Chargebacks and Fraud →

Did this page help you? Yes No

Chargebacks and Fraud

https://docs.fatzebra.com/docs/chargebacks-and-fraud

Chargebacks and Fraud

Chargebacks - how they work and how to avoid them

What's a Chargeback?

A chargeback is like a refund. It is a right which may be exercised in certain situations by a cardholder's financial services provider against a merchant, to charge back responsibility for a card transaction from the cardholder to the merchant.

When do Chargebacks occur?

It occurs when a cardholder disputes a transaction on their credit or debit card, or through a direct debit, and asks for the charge to be reversed.

Chargebacks can only be made for certain reasons, and will depend on matters such as:

- The terms and conditions of the credit or debit card or the bank account, which will explain when and how a cardholder can claim a chargeback.
- The rules of credit or debit card scheme, such as Visa, MasterCard, or American Express.
- The Merchant Agreement, which sets out when a transaction is invalid or unacceptable, and is liable to be charged back to the merchant (where the dispute is lodged by a merchant)
- Reasons given by the cardholder for wanting the transaction charged back (where the disputes is lodged by the cardholder).

If the cardholder's reason for disputing the transaction does not fall within the card scheme's reasons, they may not be able to reverse the transaction. In these circumstances, the cardholder may need to take up their complaint directly with the merchant (the company that provided the goods or service).

It is important to remember that the cardholder's bank is required to claim a chargeback and the merchant's bank is obliged to process a chargeback against the merchant's facility if the reason for the chargeback is consistent with the relevant card scheme rules. Some common reasons for chargebacks are:

- No cardholder authorisation
- Illegal transaction
- Forged signature on voucher
- Expired card
- Goods/services not supplied
- Transaction over floor limit without authorisation
- Processing offline when terminal working

A merchant should respond promptly to any chargeback request and supply relevant information to its FSP.

Most merchant chargeback disputes are raised where a merchant has a chargeback claimed against them or where the merchant is a victim of fraud and the dispute is lodged against the merchant.

Updated over 6 years ago

← Remittance Merchants

Supported Currencies →

Did this page help you? Yes No

Supported Currencies

https://docs.fatzebra.com/docs/supported-currencies

Supported Currencies

See what currencies are supported by Cloud Payments

Fat Zebra supports processing payments in 100+ currencies. This is especially helpful if you have a global presence, as charging in a customer's native currency can increase sales.

The currency used comes into play in three places:

- The customer's credit card currency
- The currency of the charge, called the presentment currency
- The currency accepted by your destination bank account or debit card, called the settlement currency

If the charge currency differs from the customer's credit card currency, the customer may be charged a foreign exchange fee by their credit card company. The customer may also be charged a fee by their credit card company if the credit card and your business are in different countries, regardless of the currency used.

> Please ensure your acquirer supports the currencies you require.

Updated over 4 years ago

← Chargebacks and Fraud

ISO Currency Codes →

Did this page help you? Yes No

ISO Currency Codes

https://docs.fatzebra.com/docs/iso-currency-codes

ISO Currency Codes

Below is a list of ISO 4217 Currency Codes

> The three-letter ISO code is provided for each currency below, but you should provide the ISO code in all UPPERCASE letters when making the Purchase Request

Currency	Alphabetic Code	Minor Unit
Afghani	AFN	2
Algerian Dinar	DZD	2
Argentine Peso	ARS	2
Armenian Dram	AMD	2
Aruban Florin	AWG	2
Australian Dollar	AUD	2
Azerbaijan Manat	AZN	2
Bahamian Dollar	BSD	2
Bahraini Dinar	BHD	3
Baht	THB	2
Balboa	PAB	2
Barbados Dollar	BBD	2
Belarusian Ruble	BYN	2
Belize Dollar	BZD	2
Bermudian Dollar	BMD	2
Bolívar	VEF	2
Boliviano	BOB	2
Brazilian Real	BRL	2
Brunei Dollar	BND	2
Bulgarian Lev	BGN	2
Burundi Franc	BIF	0
Cabo Verde Escudo	CVE	2
Canadian Dollar	CAD	2
Cayman Islands Dollar	KYD	2
CFA Franc BCEAO	XOF	0
CFP Franc	XPF	0
Chilean Peso	CLP	0
Codes specifically reserved for testing purposes	XTS	N.A.
Colombian Peso		
Comorian Franc	COP	2
Congolese Franc	KMF	0
Convertible Mark	CDF	2
Cordoba Oro	BAM	2
Costa Rican Colon	NIO	2
Cuban Peso	CRC	2
Czech Koruna	CUP	2
Dalasi	CZK	2
Danish Krone	GMD	2
Denar	DKK	2
Djibouti Franc	MKD	2
Dobra	DJF	0
Dominican Peso	STN	2
Dong	DOP	2
East Caribbean Dollar	VND	0
Egyptian Pound	XCD	2
El Salvador Colon	EGP	2
Ethiopian Birr	SVC	2

Currency	Alphabetic Code	Minor Unit
Euro	ETB	2
Falkland Islands Pound	EUR	2
Fiji Dollar	FKP	2
Forint	FJD	2
Ghana Cedi	HUF	2
Gibraltar Pound	GHS	2
Gold	GIP	2
Gourde	XAU	N.A.
Guarani	HTG	2
Guinean Franc	PYG	0
Guyana Dollar	GNF	0
Hong Kong Dollar	GYD	2
Hryvnia	HKD	2
Iceland Krona	UAH	2
Indian Rupee	ISK	0
Iranian Rial	INR	2
Iraqi Dinar	IRR	2
Jamaican Dollar	IQD	3
Jordanian Dinar	JMD	2
Kenyan Shilling	JOD	3
Kina	KES	2
Kuna	PGK	2
Kuwaiti Dinar	HRK	2
Kwanza	KWD	3
Kyat	AOA	2
Lao Kip	MMK	2
Lari	LAK	2
Lebanese Pound	GEL	2
Lek	LBP	2
Lempira	ALL	2
Leone	HNL	2
Liberian Dollar	SLL	2
Libyan Dinar	LRD	2
Lilangeni	LYD	3
Loti	SZL	2
Malagasy Ariary	LSL	2
Malawi Kwacha	MGA	2
Malaysian Ringgit	MWK	2
Mauritius Rupee	MYR	2
Mexican Peso	MUR	2
Mexican Unidad de Inversion (UDI)	MXN	2
Moldovan Leu	MXV	2
Moroccan Dirham		
Mozambique Metical	MDL	2
Mvdol	MAD	2
Naira	MZN	2
Nakfa	BOV	2
Namibia Dollar	NGN	2
Nepalese Rupee	ERN	2
Netherlands Antillean Guilder	NAD	2
New Israeli Sheqel	NPR	2
New Taiwan Dollar	ANG	2
New Zealand Dollar	ILS	2
Ngultrum	TWD	2
North Korean Won	NZD	2
Norwegian Krone	BTN	2
Ouguiya	KPW	2
Pa'anga	NOK	2
Pakistan Rupee	MRU	2
Palladium	TOP	2
Pataca	PKR	2
Peso Convertible	XPD	N.A.
Peso Uruguayo	MOP	2
Philippine Piso	CUC	2
Platinum	UYU	2
Pound Sterling	PHP	2
Pula	XPT	N.A.
Qatari Rial	GBP	2
Quetzal	BWP	2
Rand	QAR	2

Currency	Alphabetic Code	Minor Unit
Rial Omani	GTQ	2
Riel	ZAR	2
Romanian Leu	OMR	3
Rufiyaa	KHR	2
Rupiah	RON	2
Russian Ruble	MVR	2
Rwanda Franc	IDR	2
Saint Helena Pound	RUB	2
Saudi Riyal	RWF	0
Serbian Dinar	SHP	2
Seychelles Rupee	SAR	2
Silver	RSD	2
Singapore Dollar	SCR	2
Sol	XAG	N.A.
Solomon Islands Dollar	SGD	2
Som	PEN	2
Somali Shilling	SBD	2
Somoni	KGS	2
South Sudanese Pound	SOS	2
Sri Lanka Rupee	TJS	2
Sucre	SSP	2
Sudanese Pound	LKR	2
Surinam Dollar	XSU	N.A.
Swedish Krona	SDG	2
Swiss Franc	SRD	2
Syrian Pound	SEK	2
Taka	CHF	2
Tala	SYP	2
Tanzanian Shilling	BDT	2
Tenge	WST	2
Trinidad and Tobago Dollar	TZS	2
Tugrik	KZT	2
Tunisian Dinar	TTD	2
Turkish Lira	MNT	2
Turkmenistan New Manat	TND	3
UAE Dirham	TRY	2
Uganda Shilling	TMT	2
Unidad de Fomento	AED	2
Unidad de Valor Real	UGX	0
Uruguay Peso en Unidades	CLF	4
US Dollar	COU	2
Uzbekistan Sum	UYI	0
Vatu	USD	2
WIR Euro	UZS	2
WIR Franc	VUV	0
Won	CHE	2
Yemeni Rial	CHW	2
Yen	KRW	0
Yuan Renminbi	YER	2
Zambian Kwacha	JPY	0
Zimbabwe Dollar	CNY	2
Zloty	ZMW	2
	ZWL	2
	PLN	2

Updated over 6 years ago

← Supported Currencies

AVS →

Did this page help you? Yes No

AVS

https://docs.fatzebra.com/docs/avs

AVS

> 📘 **Limited Acquirer Support**
>
> AVS checks are currently only supported by a limited number of acquirers. If the AVS fields are used where support is not yet available, the transactions will process as normal and return an `avs_result_code` of `101` which indicates that AVS is not supported.

AVS (Address Verirfication System) is a service that combats fraudulent activity by verifying a cardholder's address information against the card issuer's records.

AVS is currently supported by Fat Zebra on Purchases via the `billing_address` field. If this field is specified, the Fat Zebra API will return an `avs_result_code` field indicating the result of the AVS check.

The following `avs_result_code` values may be returned:

avs_result_code	Meaning
`0`	Both address and postcode match
`1`	Both address and postcode do not match
`2`	Address matches, postcode does not match
`3`	Address matches, postcode not verified
`4`	Postcode matches, address noes not match
`5`	Postcode matches, address not verified
`6`	Both address and postcode not verified
`7`	Postcode matches
`8`	Cardholder name, address, and postcode match
`9`	Cardholder name, address, and postcode does not match
`10`	Both cardholder name and address match
`11`	Both cardholder name and postcode match
`12`	Cardholer name matches
`13`	Address and postcode matches, cardholder name does not match
`14`	Address matches, cardhodler name does not match
`15`	Postcode matches, cardholder name does not match
`100`	AVS unavailable
`101`	AVS not supported
`102`	Address information is unavailable

Updated over 4 years ago

← ISO Currency Codes

Merchant Advice Codes (Retries) →

Did this page help you? Yes No

Merchant Advice Codes (Retries)

https://docs.fatzebra.com/docs/merchant-advice-codes-retries

Merchant Advice Codes (Retries)

For declined transactions there is a field returned advising the merchant how they should process a retry. The possible values and explanations is listed below:

Code	Description/Action to take
update_required	The merchant should contact the customer to obtain updated payment method details. This may be for a new expiry date, CVV or similar.
retry_later	The merchant should retry the transaction after the timestamp provided in `merchant_advice_retry_after` .
do_not_retry	Do not retry the payment. Contact the customer to obtain an alternative payment method for this transaction.
not_supported	The payment token used is not supported for this transaction. Contact the customer to obtain an alternative payment method for this transaction.
cancelled	The recurring agreement has been cancelled by the customer - Do Not Retry. Contact the customer if necessary to establish a new agreement.
none	No advice code has been provided. Retry processing up to the merchant to determine.

Retry Later

If a `retry_later` code is received, the merchant should retry after the timestamp provided in the `merchant_advice_retry_after` field. If the field is absent, assume 72 hours.

JSON

```
{
   // ...
   "merchant_advice_code": "retry_later",
   "merchant_advice_retry_after": "2022-10-04T15:39:55+11:00"
}
```

Updated about 2 years ago

← AVS

Tokenized Credit Cards →

Did this page help you? Yes No

Tokenized Credit Cards

https://docs.fatzebra.com/docs/tokenized-credit-cards

Tokenized Credit Cards

Tokenized Credit Cards is a method of storing your customers credit card details securely 'on file' in the Fat Zebra secure cardholder data environment.

Commonly this is used for recurring payments, however this can also be used to provide a streamlined customer experience, allowing them to save their payment details for future orders.

When a Credit Card is tokenized, the Gateway will return a token that is unique to your merchant account. To charge the Tokenized Credit Card, you should provide the card token in lieu of the other card details, and this is used when authorizing the payment.

Every Credit Card transaction within the Gateway results in the tokenization of a Credit Card, and the token is returned with every purchase request. This allows merchants to store the token after an order has been placed, or after an order has been processed in an out of band payment method (such as though Direct Post or a Hosted Payment Page).

Merchants can assign (if desired) an alias to the Tokenized Credit Card, which allows for a custom value to be associated with the Token for use when processing the payment. This allows for a merchant customer reference (such as a customer ID or number) to be used instead of the system generated token value.

Should a Tokenized Credit Card expire, you can update the record on file with a new expiry date when it is provided by the customer.

You can find more details on how to tokenize a credit card here.

Transact With Tokenized Card Example

In order to transact with a tokenized credit card, you will need to tokenize the credit card first.

Tokenize Credit Card

```
/*
 * Request: POST /v1.0/credit_cards
 */
{
  "card_number": "5123456789012346",
  "card_holder": "Bob Smith",
  "card_expiry": "05/2030",
  "cvv": "987"
}
```

You will get a tokenized credit card in the response from `token`. In future transactions, you can just transact with this token instead of using the credit card.

Tokenize Credit Card Response

```
/*
 * Response
 */
{
  "successful": true,
  "response": {
    "token": "fke8jmra",
    "card_holder": "Bob Smith",
    "card_number": "512345XXXXXX2346",
    "card_expiry": "2030-05-31",
    "card_type": "MasterCard",
    "card_category": "Credit",
    "card_subcategory": "Standard",
    "card_issuer": "Banco Del Pichincha, C.A.",
    "card_country": "Ecuador",
    "authorized": true,
    "transaction_count": 0,
```

```
    "alias": null
  },
  "errors": [],
  "test": true
}
```

With your tokenized credit card, you can now transact with the card.

Transact With Tokenized Credit Card

```
/*
 * Request: /v1.0/purchases
 */
{
  "amount": 100,
  "currency": "AUD",
  "card_token": "fke8jmra",
  "reference": "transact_with_token"
}
```

Updated over 3 years ago

← Merchant Advice Codes (Retries)

Recurring / Installment Payments →

Did this page help you? Yes No

Recurring / Installment Payments

https://docs.fatzebra.com/docs/recurring-instalment-payments

Recurring / Installment Payments

Merchants can make recurring and installment transactions. This is achieved by:

1. Set `ECM` value accordingly
2. Set `stored_credential_indicator` to indicate if it's the first or subsequent transaction.
3. Using the `card_token` from the initial transaction where the customer authorized the recurring or installment cycle, as well as specifying the `authorization_tracking_id` returned on the initial transaction in all subsequent recurring or installment transactions.
4. If the card is expired, you can set `omit_expiry` to true to allow payment to go through.

The definition of `ECM`, `card_on_file` and `omit_expiry` can be found here.

A Recurring Scenario Example

Below is a scenario example to illustrate how to use `card_token` and `authorization_tracking_id` for first and subsequent transactions:

First Recurring Transaction

We can create the first recurring transaction via create a purchase.

First Recurring Payment

```
/*
 * Request: POST /v1.0/purchases
 */
{
  "amount": 100,
  "currency": "AUD",
  "card_holder": "John Smith",
  "card_number": "411111xxxxxx1111",
  "card_expiry": "01/2025",
  "cvv": "123",
  "reference": "first_recurring_transaction",
  "extra": {
    "ecm": 32,
    "stored_credential_indicator": "I"
  }
}
```

In the response, we will get the `card_token` as well as the `authorization_tracking_id`. These values will need to be stored for the subsequent transaction.

- `card_token`: this is a token for the card number used in the initial transaction and can be used in subsequent transactions so that the actual card number does not need to be entered
- `metadata.authorization_tracking_id`: this is an ID returned by the card schemes which must be presented on subsequent recurring or installment transactions.

First Recurring Payment Response

```
/*
 * Response
 */
{
  "successful": true,
  "response": {
    "authorization": "123456",
    "id": "1-P-VBF4KCE9",
    "card_number": "411111xxxxxx1111",
    "card_holder": "John Smith",
```

```
    "card_expiry": "2025-01-31",
    "card_token": "lk78b0xadcdek2hffvkadif",
    "card_type": "VISA",
    "card_category": "Credit",
    "card_subcategory": "Standard",
    "amount": 100,
    "decimal_amount": 1.0,
    "successful": true,
    "message": "approved - balance avail",
    "reference": "first_recurring_transaction",
    "currency": "AUD",
    "transaction_id": "1-P-VBF4KCE9",
    "settlement_date": "2020-04-23",
    "transaction_date": "2020-04-23T10:21:31+10:00",
    "response_code": "00",
    "captured": true,
    "captured_amount": 100,
    "rrn": "042321000062",
    "cvv_match": "M",
    "metadata": {
      "authorization_tracking_id": "HNR0003960518",
      "card_sequence_number": "",
      "sca_exemption": "",
      "original_transaction_reference": ""
    },
    "addendum_data": {}
  },
  "errors": [],
  "test": false
}
```


Subsequent Transaction

The `authorization_tracking_id` should be supplied for every subsequent recurring transaction to increase approval rates.

> 📘 Some schemes do not return an `authorization_tracking_id` after a first recurring payment. If this is not present in the payload, it's safe to create recurring transactions without it.

Subsequent Recurring Transaction


```
/*
 * Request: POST /v1.0/purchases
 */
{
  "amount": 100,
  "currency": "AUD",
  "card_token": "lk78b0xadcdek2hffvkadif",
  "reference": "second_recurring_transaction",
  "extra": {
    "ecm": 32,
    "authorization_tracking_id": "HNR0003960518",
    "stored_credential_indicator": "S"
  }
}
```


Transact With Expired Tokenized Credit Card

If the tokenized credit card is expired, it will likely be rejected. You can set `omit_expiry:true` to attempt to increase approval rates.

> 📘 **Automatic Card Updates**
>
> Contact your account manager to enquire about Automatic Card Update capabilities (including automatic expiry date updates) to help uplift approval rates for expired cards.

Transact With Expired Tokenized Credit Card


```
/*
 * Request: POST /v1.0/purchases
 */
{
  "amount": 100,
  "currency": "AUD",
  "card_token": "lk78b0xadcdek2hffvkadif",
  "reference": "second_recurring_transaction",
```

```
  "extra": {
    "ecm": 32,
    "omit_expiry": true,
    "authorization_tracking_id": "HNR0003960518",
    "stored_credential_indicator": "S"
  }
}
```

Updated over 1 year ago

← Tokenized Credit Cards

Merchant Initiated Transaction →

Did this page help you? Yes No

Merchant Initiated Transaction

https://docs.fatzebra.com/docs/merchant-initiated-transaction

Merchant Initiated Transaction

A merchant initiated transaction is a payment that is agreed with payer's consent, and is initiated by the merchant collecting the payment.

In order to apply for a merchant initiated transaction, the merchant needs to specify why this transaction is initiated, as well as the `authorization_tracking_id` of the original transaction, and `stored_credential_indicator` to show if it's the first card on file transaction on the subsequent one.

You can see different reasons here.

A Merchant Initiated Transaction Example

In order to apply a merchant initiated transaction, the customer will need to put their card on file first.

This transaction can be a zero dollar authorization just to put card on file, or it could be the initial payment customer is required to pay. We will assume customer is just doing a zero dollar authorization. We can create the transaction via Create a purchase .

Note that since we are putting card on file, we will need to specify `card_on_file` to true.

Customer Initiated Transaction

```
/*
 * Request: POST /v1.0/purchases
 */
{
  "amount": 0,
  "currency": "AUD",
  "card_holder": "John Smith",
  "card_number": "411111xxxxxx1111",
  "card_expiry": "01/2025",
  "cvv": "123",
  "reference": "customer_initiated_transaction",
  "extra": {
    "card_on_file": true,
    "stored_credential_indicator": "I"
  }
}
```

In the response, we will get the `card_token` as well as the `authorization_tracking_id` . These values will need to be stored for the merchant initiated transaction later.

Customer Initiated Transaction Response

```
/*
 * Response
 */
{
  "successful": true,
  "response": {
    "authorization": "123456",
    "id": "1-P-VBF4KCE9",
    "card_number": "411111xxxxxx1111",
    "card_holder": "John Smith",
    "card_expiry": "2025-01-31",
    "card_token": "lk78b0xadcdek2hffvkadif",
    "card_type": "VISA",
    "card_category": "Credit",
    "card_subcategory": "Standard",
    "amount": 100,
    "decimal_amount": 1.0,
    "successful": true,
```

```
    "message": "approved - balance avail",
    "reference": "first_recurring_transaction",
    "currency": "AUD",
    "transaction_id": "1-P-VBF4KCE9",
    "settlement_date": "2020-04-23",
    "transaction_date": "2020-04-23T10:21:31+10:00",
    "response_code": "00",
    "captured": true,
    "captured_amount": 100,
    "rrn": "042321000062",
    "cvv_match": "M",
    "metadata": {
      "authorization_tracking_id": "HNR0003960518",
      "card_sequence_number": "",
      "sca_exemption": "",
      "original_transaction_reference": ""
    },
    "addendum_data": {}
  },
  "errors": [],
  "test": false
}
```


To apply for a merchant initiated transaction, you need to apply the `card_token`, the `auth_reason` and `authorization_tracking_id` in the payload.

Note: Not all schemes requires `authorization_tracking_id`. If this value is not found in the response, then it indicates this scheme does not support it, and this can be ignored for subsequent transactions.

> ❗ Failure to present the authorization_tracking_id may result in the recurring or installment transaction being declined, particularly wallet transactions.

JSON

```
{
  "amount": 100,
  "currency": "AUD",
  "card_token": "lk78b0xadcdek2hffvkadif",
  "reference": "merchant_initiated_transaction",
  "extra": {
    "auth_reason": "no_show",
    "authorization_tracking_id": "HNR0003960518",
    "stored_credential_indicator": "S"
  }
}
```

Updated 3 months ago

← Recurring / Installment Payments

Incremental Authorization →

Did this page help you? Yes No

Incremental Authorization

https://docs.fatzebra.com/docs/incremental-authorization

Incremental Authorization

Merchant can make incremental authorization to change the amount of authorization in the lifecycle of the authorization.

There are two types of incremental authorization:

1. Top up the amount - This increases the amount of the pre-authorization, and extend the chargeback expiry date.
2. Update the pre-authorization expiry date - This only extends the pre-authorization expiry date.

Note:

- By default, pre-authorization stays for 7 days, and every extension will extend it by 7 days.
- Extension can only be done before it is expired.

Incremental Auth Example

Below is a scenario example to illustrate how to do a top up and extend the chargeback expiry date

Pre-authorization Transaction

You can reference Create an authorization for more details. To show it is a pre-authorization, set `capture: false`.

Pre-authorization

```
/*
 * Request: POST /v1.0/purchases
 */
{
  "amount": 100,
  "currency": "AUD",
  "card_holder": "John Smith",
  "card_number": "411111xxxxxx1111",
  "card_expiry": "01/2025",
  "cvv": "123",
  "reference": "first_preauth_transaction",
  "capture": false
}
```

Pre-authorization Response

```
/*
 * Response
 */
{
  "successful": true,
  "response": {
    "authorization": "123456",
    "id": "1-P-VBF4KCE9",
    "card_number": "411111xxxxxx1111",
    "card_holder": "John Smith",
    "card_expiry": "2025-01-31",
    "card_token": "lk78b0xadcdek2hffvkadif",
    "card_type": "VISA",
    "card_category": "Credit",
    "card_subcategory": "Standard",
    "amount": 100,
    "decimal_amount": 1.0,
    "successful": true,
    "message": "approved - balance avail",
    "reference": "first_preauth_transaction",
    "currency": "AUD",
    "transaction_id": "1-P-VBF4KCE9",
```

```
    "settlement_date": "2020-04-23",
    "transaction_date": "2020-04-23T10:21:31+10:00",
    "response_code": "00",
    "captured": false,
    "captured_amount": 100,
    "rrn": "042321000062",
    "cvv_match": "M",
    "metadata": {
      "authorization_tracking_id": "HNR0003960518",
      "card_sequence_number": "",
      "sca_exemption": "",
      "original_transaction_reference": ""
    },
    "addendum_data": {}
  },
  "errors": [],
  "test": false
}
```


Top-up Transaction (incremental auth)

To top up the transaction, you have to provide an amount that is bigger than the original amount. In this case, we need to provide a value bigger than 100. Let's assume we need to increment the transaction by $1.00, then we should send 200. See more details in Update an authorization

Note that you will need to provide the previous response ID "1-P-VBF4KCE9" in the url.

Top Up Payment

```
/*
 * Request: PUT /v1.0/purchases/1-P-VBF4KCE9
 */
{
  "amount": 200
}
```

Now the pre-authorization is extended to $2.00, and the new expiry date is 7 days after this transaction time.

Top Up Payment Response


```
{
  "successful": true,
  "response": {
    "authorization": "123456",
    "id": "1-P-VBF4KCE9",
    "card_number": "411111xxxxxx1111",
    "card_holder": "John Smith",
    "card_expiry": "2025-01-31",
    "card_token": "lk78b0xadcdek2hffvkadif",
    "card_type": "VISA",
    "card_category": "Credit",
    "card_subcategory": "Standard",
    "amount": 200,
    "decimal_amount": 2.0,
    "successful": true,
    "message": "Approved",
    "reference": "first_preauth_transaction",
    "currency": "AUD",
    "transaction_id": "1-P-VBF4KCE9",
    "settlement_date": null,
    "transaction_date": "2020-04-26T11:13:49+10:00",
    "response_code": "00",
    "captured": false,
    "captured_amount": 0,
    "rrn": "042321000062",
    "cvv_match": "U",
    "metadata": {
      "sca_exemption": "",
      "card_sequence_number": "",
      "authorization_tracking_id": "HNR0003960518",
      "original_transaction_reference": "",
      "original_amount": "100"
    },
    "addendum_data": {}
  },
  "errors": [],
  "test": false
}
```

You will notice that in the response, the authorization is being kept the same.

Pre-authorization Expiry Extension

To extend the expiry date while keeping the amount the same, you should use `extend: true`.

Preauthorization Extension

```
/*
 * Request: PUT /v1.0/purchases/1-P-VBF4KCE9
 */
{
  "extend": true
}
```

This will allow the pre-authorization to be extended by 7 days from the day this transaction is sent.

> ! **"Amount" and "Extend" cannot be sent at the same time**
>
> When "amount" is provided in the payload, the expiry date of the pre-authorization will be extended, so you should not send "extend" if you are already doing a top up.

Pre-authorization Completion

Once the payment is ready to settle, you should send pre-authorization completion to settle the payment. This has to be done before the pre-authorization expires. See more details in Capture an authorization .

Pre-authorzation Completion (Capture Payment)

```
/*
 * Request: POST /v1.0/purchases/1-P-VBF4KCE9/capture
 */
{
  "amount": 200
}
```

Updated over 3 years ago

← Merchant Initiated Transaction

Overview →

Did this page help you? Yes No

Overview

https://docs.fatzebra.com/docs/direct-entry-overview

Overview

Direct Entry (DE) payments are a method of receiving or sending money directly from or to a customers bank account. Most merchants will require Direct Debit payments, however in some cases Direct Credit is also used.

In order to make Direct Debit or Direct Credit payments, a Direct Entry (DE) facility is required. This can usually be established through your financial institution, or through a payment services provider such as Fat Zebra.

In addition to stand-alone Direct Debit payments, it is also possible to process Batch Payments, which are especially useful for large payment runs.

Fat Zebra can support debiting bank accounts in Australia and New Zealand.

Direct Entry Facility

If you would like to make Direct Debit or Direct Credit payments and already have a Direct Entry (DE) facility, you will need to provide us with the following:

- Your Direct Entry User ID (DE ID)
- Your institution code (such as WBC)
- Your User Preferred Specification (UPS) - commonly known as the display name
- Your BSB and Account Number for the source/destination account

Asynchronous Processing

Direct Entry processing is an asynchronous payment method, which means that this is not a real-time process due to the way that the banks process the payment files. Completion and rejection cannot be guaranteed until after 3 business days, as this is dependent on the network of banks to return payment statuses back to the originating bank.

Handling Rejections

The status of a direct debit can take up to 3 business days to be returned from the account holder's bank. We recommend merchants set up a webhook to receive notifications when direct debits are updated to rejected. Please see our Webhooks Overview for more information.

Please note that Direct Debit batches allow 3 business days for all rejections to be received before they are available to be downloaded.

Direct Entry Statuses

There are five states a direct entry record can be in:

- New - the record has been created in Fat Zebra's system and has not been submitted to the financial institution
- Pending - the record has been submitted to the financial institution for processing
- Completed - the record has been successfully processed
- Rejected - the record was rejected - check the result field for details
- Delete - the record has been deleted and was not submitted for processing

Testing

In the sandbox environment, the cent-amount of a direct entry can be used to simulate a specific outcome. The possible results are as follows:

Amount	Result
$x.01	Invalid BSB Number

Amount	Result
$x.02	Payment Stopped
$x.03	Account Closed
$x.04	Customer Deceased
$x.05	No Account/Incorrect Account Number
$x.06	Refer to Customer
$x.08	Invalid User ID number (configuration error with your DE facility)
$x.09	Technically Invalid

Any other value will result in a successfully completed DE.

Updated almost 2 years ago

← Incremental Authorization

Direct Debits →

Did this page help you? Yes No

Direct Debits

https://docs.fatzebra.com/docs/direct-debits-1

Direct Debits

Direct Debit is the process of debiting money from a customer's bank account through the asynchronous Direct Entry batch process.

In order to take Direct Debit payments, you must establish a Direct Debit facility through your bank or through a payment services provider like Fat Zebra. Your account manager can assist with this if required.

To debit a customer, you will require:

- The Direct Debit facility enabled for your merchant account
- The customer's BSB
- The customer's Account Number
- The name of the customer's account

Direct Debits are processed in a nightly batch with the cutoff for payments being 5PM, UTC+10 (or UTC+11 during Australian Daylight Saving Time).

Once a Direct Debit has been submitted for processing, the results will be updated overnight. However it is important to remember that it can take up to 3 business days from the date of the Direct Debit for rejections to come through from the banking network.

Due to the asynchronous nature of this payment method, it is recommended that you implement Webhooks so that your system is updated upon the completion or rejection of payments.

Updated over 4 years ago

← Overview

Direct Credits →

Did this page help you? Yes No

Direct Credits

https://docs.fatzebra.com/docs/direct-credits-1

Direct Credits

Direct Credit is the process of crediting (sending) money from your bank account (the funding account or source account) to a customer's bank account (the destination account), through the asynchronous Direct Entry batch process. Usually this is used for the distribution/disbursal of funds, reimbursements, etc.

In order to send Direct Credit payments, you must establish a Direct Credit facility through your bank. Your account manager can assist with this if required.

To credit funds to a customer, you will require:

- The Direct Credit facility enabled for your merchant account
- The customer's BSB
- The customer's Account Number
- The name of the customer's account

Direct Credits are processed in a nightly batch, with the cutoff for payments being 5PM, UTC+10 (or UTC+11 during Australian Daylight Saving Time). If there is insufficient funds to process the whole file submitted to the bank, your Direct Credits will fail and will be updated accordingly.

Once a Direct Credit has been submitted for processing, the funds will arrive in the destination account within 24 hours, usually the next business day.

Due to the asynchronous nature of this payment method, it is recommended that you implement Webhooks so that your system is updated upon the completion or rejection of payments.

Updated about 6 years ago

← Direct Debits

Overview →

Did this page help you? Yes No

Overview

https://docs.fatzebra.com/docs/overview

Overview

Card purchases and bank account direct debits can be processed in bulk by uploading a batch file.

File Name

The file name must use the following format: `[Prefix]-[Version]-[Type]-[Merchant Username]-[Date]-[Suffix].csv`
For example: `BATCH-v1-PURCHASE-acmeinc-20180131-1516337063.csv`

Filename Component	Example Value	Description
Prefix	BATCH	This value must always be "BATCH". It indicates that this is a batch processing file.
Version	v1	The version of the batch file. This should currently be set to "v1".
Type	PURCHASE	This indicates the type of records in the file. Use "PURCHASE", "REFUND" or "DIRECTDEBIT"
Merchant Username	acmeinc	Your merchant username.
Date	20180131	The desired processing date for the batch. If the date provided is current or past, the batch will be queued for immediate processing.
Suffix	1516337063	Choose your own suffix for distinguishing between batches scheduled on the same date (e.g. file created timestamp in epoch time). This should be no longer than 15 characters.

File Format Requirements

The batch file must be in CSV format and use ASCII as the character set.

Each line must end in Carriage Return and Linefeed (CRLF).

The file must not contain any header or footer rows.

The data in the file must be text-types; i.e. if creating rows in Excel, ensure all cell types are set to Text to ensure numbers are not converted to scientific etc.

Validation

When a batch file is received for processing it will be validated and then enqueued to be processed on the batch date (or immediately if the date is the same as the upload date or a past date).

Processing & Completion

For batches which need to be processed as soon as they are received, the date in the filename should be set to the current date. If a batch is for a future date then this date should be used in the filename.

Batches which are for immediate processing will be started as soon as they are properly validated, however they may be queued behind other batches. For future dated batches, processing will commence at midnight, UTC+10 (or UTC+11 during AEDT) (14:00 UTC).

A results file can be downloaded which details the outcome of each transaction.

Updated over 1 year ago

← Direct Credits

Purchase Batch File Columns →

Did this page help you? Yes No

Purchase Batch File Columns

https://docs.fatzebra.com/docs/purchase-batch-specification

Purchase Batch File Columns

Field Name	Example Value	Description
Amount	10050 Integer Required	The amount of the transaction to be processed. This is an integer in the smallest units for the currency (i.e. $100.50 will be 10050)
Currency	AUD String (3 characters) Required	The ISO Currency code for the transaction.
Reference	INV-1234 String (100 characters) Required	The merchant reference for the transaction. This must be unique. It is recommended that this is alphanumeric only with the exception of hyphen (-) and underscore (_)
Card Holder	Max Smith String (50 characters) Required unless Token is present	The card holder's name.
Card Number	4444333322221111 String (19 characters) Required unless Token is present	The card number.
Card Expiry	09/2023 Date (mm/yyyy) Required unless Token is present	The card expiry date in mm/yyyy format.
Token	a348jki String (20 characters) Required unless Card Holder, Card Number and Card Expiry are present.	The card token to be used in place of Card Holder, Card Number and Card Expiry. This column takes priority over the other card detail columns.
Description	Invoice for EOFY accounts String (255 characters) Optional	The description for the transaction.

Updated about 6 years ago

← Overview

Refund Batch File Columns →

Did this page help you? Yes No

Refund Batch File Columns

https://docs.fatzebra.com/docs/refund-batch-file-columns

Refund Batch File Columns

Field Name	Example Value	Description
Amount	10050 Integer Required	The amount of the transaction to be processed. This is an integer in the smallest units for the currency (i.e. $100.50 will be 10050).
Currency	AUD String (3 characters) Required	The ISO Currency code for the transaction.
Reference	INV-1234 String (100 characters) Required	The merchant reference for the transaction. This must be unique. It is recommended that this is alphanumeric only with the exception of hyphen (-) and underscore (_).
Card Holder	Max Smith String (50 characters) Required unless Token or Purchase ID is present	The card holder's name.
Card Number	4444333322221111 String (19 characters) Required unless Token or Purchase ID is present	The card number.
Card Expiry	09/2023 Date (mm/yyyy) Required unless Token or Purchase ID is present	The card expiry date in mm/yyyy format.
Token	a348jki String (20 characters) Required unless Purchase ID is present or unless Card Holder, Card Number and Card Expiry are present.	The card token to be used in place of Card Holder, Card Number and Card Expiry. This column takes priority over the other card detail columns.
Description	Invoice for EOFY accounts String (255 characters) Optional	The description for the transaction.
Purchase ID	071-P-ABCD1234 String (15 characters) Required unless Token is present or unless Card Holder, Card Number and Card Expiry are present.	If refunding a previous purchase, this is the ID of the purchase that should be refunded.

Updated over 5 years ago

← Purchase Batch File Columns

Direct Debit Batch File Columns →

Did this page help you? Yes No

Direct Debit Batch File Columns

https://docs.fatzebra.com/docs/direct-debit-batches

Direct Debit Batch File Columns

Field Name	Example Value	Description
Amount	10050 Integer Required	The amount of the transaction to be processed. This is an integer in the smallest units for the currency (i.e. $100.50 will be 10050).
Currency	AUD String (3 characters) Required	The ISO Currency code for the transaction. Currently only AUD is supported for Direct Debit.
Reference	INV-1234 String (64 characters) Required	The merchant reference for the transaction. This must be unique. It is recommended that this is alphanumeric only with the exception of hyphen (-) and underscore (_).
Account Name	Max Smith String (32 characters) Required unless Bank Account ID is present	The bank account holder's name.
BSB	633-000 String (7 characters) Required unless Bank Account ID is present	The bank BSB in the ###-### format.
Account Number	112498233 String (9 characters, numeric) Required unless Bank Account ID is present	The bank account number, maximum 9 digits.
Bank Account ID	071-BA-AZ7JK98L String (20 characters) Required unless Account Name, BSB and Account Number are present.	The Bank Account ID to be used for the transaction instead of the bank account details. This column takes priority over the other bank account columns.
Description	Invoice INV-1234 String (18 characters) Required	The description for the transaction. This will appear on the customer's bank statement and must be unique.

Updated about 6 years ago

Refund Batch File Columns

Result Files

Did this page help you? Yes No

Result Files

https://docs.fatzebra.com/docs/result-files

Result Files

Result File Contents

Upon completion of a batch the result file can be retrieved to be ingested into other systems.

Result file contents for PURCHASE/REFUND batches

Column Name	Example Value	Description
Reference	INV-1234 String (100 characters)	The merchant reference for the transaction.
Transaction ID	071-P-ABCD1234A String	The gateway transaction ID.
Result	Approved String	Whether the transaction was Approved or Declined.
Response Code	08 String (2 digits)	The bank response code.
Message	Approved String	The message for the bank response code.
Authorization ID	983423 String (6 digits)	The transaction authorisation ID.
RRN	348723409134 String (12 digits)	The transaction reference retrieval number (RRN, also known as receipt number).
Amount	10050 Numeric	The transaction amount.
Currency	AUD String (3 characters)	The ISO Currency code for the transaction.
Card Holder	Max Smith String (18 characters)	The card holder's name.
Card Number	444433XXXXXX1111 String (19 characters)	The masked card number for the transaction.
Card Expiry	09/2023 Date (mm/yyyy)	The card expiry date in mm/yyyy format.
Token	jksh328h String (20 characters)	The card token to be used in place of Card Holder, Card Number and Card Expiry.

Column Name	Example Value	Description
Description	Invoice INV-1234 String (32 characters)	The description for the transaction.

Result file contents for DIRECTDEBIT batch

Column Name	Example Value	Description
Reference	INV-1234 String (100 characters)	The merchant reference for the transaction.
Transaction ID	071-DD-ABCD1234A String	The gateway direct debit ID.
Result	Approved String	Whether the transaction was Approved or Declined.
Message	Approved String	The message for the bank response code.
Amount	10050 Numeric	The amount of the transaction to be processed. This is an integer in the smallest units for the currency (i.e. $100.50 will be 10050)
Currency	AUD String (3 characters)	ISO Currency code for the transaction. Currently only AUD is supported for Direct Debit.
Reference	INV-1234 String (100 characters)	The merchant reference for the transaction. This must be unique. It is recommended that this is alphanumeric only with the exception of hyphen (-) and underscore (_)
Account Name	Max Smith String (18 characters)	The bank account holders name.
BSB	633-000 String (7 characters)	The bank BSB in the ###-### format.
Account Number	112498233 String (9 characters, numerical)	The bank account number, maximum 9 digits.
Bank Account ID	071-BA-AZ7JK98L String (20 characters)	The Bank Account ID to be used for the transaction instead of the bank account details. This column takes priority over the other bank account columns.
Description	Invoice INV-1234 String (18 characters)	The description for the transaction. This will appear on the customers bank statement.

Updated over 5 years ago

← Direct Debit Batch File Columns

Upload Specification →

Did this page help you? Yes No

Upload Specification

https://docs.fatzebra.com/docs/invoices-csv

Upload Specification

The Persisted Invoices CSV file must contain five columns in the following order: reference, description, currency, amount, payable. No header row should be included - the first row should contain details of the first invoice to be imported.

Column	Type	Example	Description
reference	Alphanumeric String	abcd1234	A reference for the invoice.
description	Alphanumeric String	Doloribus qui odit fugit doloremque quod nihil. Voluptate eos sed debitis ut reiciendis distinctio consequuntur. Ab culpa dolorem iste quisquam nesciunt reiciendis deserunt.	A description for the invoice. May contain new lines.
currency	ISO 4217 Currency Code	AUD	The currency that the invoice is payable in.
amount	Integer	123456	The amount due on the invoice measured in the minor units of currency e.g. cents, pennies etc. For example AUD + 123456 = $1,234.56
payable	Boolean	true	Whether or not the invoice can be paid.

example.csv

```
410c2106,"Nostrum esse ipsa deserunt ut aut quidem eaque.
Officiis qui et optio qui quaerat quas eligendi.
Quae rerum repellat officia sunt exercitationem laboriosam.",GBP,545936,true
8f25c865,"Sit perferendis minus et.
Aut odit voluptatem laudantium molestiae ullam.
Explicabo vitae et voluptatem.",AUD,836755,false
```

Updated about 6 years ago

← Result Files

Apple Pay →

Did this page help you? Yes No

Apple Pay

https://docs.fatzebra.com/docs/apple-pay

Apple Pay

Accepting Apple Pay is faster than accepting traditional credit and debit cards and other payment methods. Customers no longer need to spend time searching for their wallet and finding the right card. Within apps or websites when using Safari, your customers can check out with a single touch.

Apple Pay Checkout Workflow

- A customer adds a credit, debit, or prepaid card into their Apple Wallet app on their iPhone, iPad, Apple Watch or Mac.
- The customer chooses to Buy with Apple Pay on the merchants website or iOS app and only needs to then select their card and shipping address.
- The customer's card details are retrieved securely by Fat Zebra and your systems are not exposed to the PCI Scope for this data.

Pre-requisites

Apple Guidelines

Before starting your Apple Pay integration, please review Apple's documentation on how to prepare your app and/or website to support Apple Pay:

- Planning For Apple Pay
- Apple Pay Acceptable Use Guidelines For Websites
- Human Interface Guidelines
- Apple Sandbox Testing

Setup your Apple Merchant ID

An Apple Merchant ID is an identifier you register with Apple that uniquely identifies your business as a merchant able to accept payments.
Follow the instructions in Setup your Apple Merchant ID for instructions to create this ID.

Implementing Apple Pay in Apps

Apple provide a sample Xcode app that implements Apple Pay that can be used for reference.

Please follow the **official documentation from Apple for implementing Apple Pay in Apps**.

In summary, you will need to implement the following to support Apple Pay in your app, with some steps requiring configuration specific to Fat Zebra:

- Enable Apple Pay in Xcode
- Determine if the device supports Apple Pay and whether the user has added payment cards using `PKPaymentAuthorizationController` and `PKPaymentAuthorizationViewController`
- Provide a button that is used either to trigger payments through Apple Pay or to prompt the user to set up a card with `PKPaymentButton`
- Display a request for payment, including information about payment processing capabilities, the payment amount, and shipping information using `PKPaymentRequest`
- Handling the callback in `PKPaymentAuthorizationControllerDelegate` that is made when the user has authorized the payment, and calling the Fat Zebra API to **create a purchase using a wallet**

Implementing Apple Pay on the Web

Please follow the **official documentation from Apple for implementing Apple Pay on the Web**.

Your implementation must meet the following pre-requisites as prescribed by Apple:

- Your website must comply with Apple Pay on the Web: Acceptable Use Guidelines
- All pages that include Apple Pay must be served over HTTPS
- Your Apple Merchant ID must have a **Merchant Identity Certificate** (which you will request from Fat Zebra) upload to it in the Apple Developer Dashboard, see Setup your Apple Merchant ID for instructions
- Any domains that will be used for Apple Pay transaction must be registered and verified in the Apple Developer Dashboard

Implementing the following to support Apple Pay on you website will require the following steps, with some steps requiring configuration specific to Fat Zebra:

- Choose an API for Implementing Apple Pay on Your Website, as you have the option of using the Apple Pay JS API or the W3C Payment Request API
- Determine if the device supports Apple Pay using `window.ApplePaySession`
- Display an Apple Pay Button on your website using CSS templates provided by Apple
- Create an Apple Pay Session & provide it a payment request
- Create an Apple Pay Session for your payment request
- Handle the `onvalidatemerchant` event and call your server passing it the URL from the event's `validationURL` property. Your server will then need to call the Fat Zebra Get Apple Pay Session endpoint. Your server will then receive an **opaque Apple Pay session object** from the Fat Zebra endpoint. This object needs to be passed onto your client side to call the Apple Pay Javascript SDK's `completeMerchantValidation` method.
- Once the user authorizes the payment, the `onpaymentauthorized` handler will be called on the `ApplePaySession`. This handler will contain a payment token encrypted by Apple using **Payment Processing Certificate**. In this event handler you can then call Fat Zebra API to **create a purchase using a wallet**, providing the payment token in the request payload

Testing in the Fat Zebra Sandbox Environment

Apple provides a Sandbox environment which allows you to test your implementation of Apple Pay with test credit and debit cards. Documentation on setting up a Apple Sandbox tester account [https://developer.apple.com/apple-pay/sandbox-testing/](can be found here).

The Fat Zebra API Sandbox environment will accept these Apple test credit and debit cards. For more information on the URL's of the Fat Zebra API Sandbox environment, refer to Endpoint Base URLs.

All Apple Pay transactions sent to the Fat Zebra API Sandbox environment will return cent-based responses, whereby the response code returned will be dependent on the amount specified in the request. E.g. for a request of $1.00, the Fat Zebra API will return a response code of "00" based on the cents of the amount. Similarly a request for $1.05 will return a response code of "05" declined. This allows you to test your integration against various response codes.

Limitations

> 🚧 **Limited Acquirer Support For Recurring Transactions**
>
> Processing recurring transactions via Apple Pay is not supported by all Australian acquirers. If these methods are used where support is not yet available the transactions may be declined or return errors.

Updated about 4 years ago

← Upload Specification

Setup your Apple Merchant ID →

Did this page help you? Yes No

Setup your Apple Merchant ID

https://docs.fatzebra.com/docs/setup-your-apple-merchant-id

Setup your Apple Merchant ID

This article provides instructions detailing how to setup an Apple Merchant ID for Apple Pay.

An Apple Merchant ID is an identifier you register with Apple that uniquely identifies your business as a merchant able to accept payments. This ID never expires, and can be used in multiple websites and iOS apps and in multiple environments (testing and production).

Request Merchant Identifier and Certificates from Fat Zebra

Contact your Fat Zebra account representative to get onboarded onto our Apple Pay integration. We will provide you with the following:

- A **Merchant Identifier** that you will need to setup your Apple Merchant ID
- A **Payment Processing Certificate** which you will need to add to the Apple Merchant ID you setup in the previous step. Apple will use this certificate to encrypt transaction data. This file will have a name ending with *-applepay.csr*.
- (*Apple Pay on the Web merchants only*) A **Merchant Identity Certificate** which you will need to upload to Apple when setting up your Apple Merchant IDs. Apple will use this certificate to authenticate your web sessions with the Apple Pay servers when using Apple Pay on the Web. This file will have a name ending with *-applepay-web.csr*.

Create an Apple Merchant ID

1. Sign in to the Apple Developer Dashboard.
2. In *Certificates, Identifiers & Profiles*, choose **iOS, tvOS, watchOS** from the pop-up menu on the left.
3. Under *Identifiers*, select **Merchant IDs**, then in the upper-right corner, click the Add button (+).
4. Enter the merchant description and identifier name provided to you by your Fat Zebra Representative.
5. Review the settings, then click **Register**.
6. Click **Done**.

Upload Your Payment Processing Certificate

1. Sign in to the Apple Developer Dashboard.
2. In *Certificates, Identifiers & Profiles*, choose **iOS, tvOS, watchOS** from the pop-up menu on the left.
3. Under *Identifiers*, select **Merchant IDs**, click on the Merchant ID that was setup in previous steps and then click **Edit**.
4. Under *Payment Processing Certificate* click on **Create Certificate**.
5. Choose **No** for the question about processing in China, then click **Continue**.
6. Click **Continue**, then **Choose File**, and select the Payment Processing CSR file provided to you by Fat Zebra (ending in *-applepay.csr*), then click **Continue**.
7. On the *Your Certificate is Ready* screen, click **Download** to save the generated certificate.
8. Send the generated certificate to your Fat Zebra account representative. Fat Zebra will load this certificate into our system, enabling your Apple Pay integration.

Upload Your Merchant Identity Certificate

You will need to upload your Merchant Identity Certificate only if you plan to support Apple Pay on the Web.

1. Sign in to the Apple Developer Dashboard.
2. In *Certificates, Identifiers & Profiles*, choose **iOS, tvOS, watchOS** from the pop-up menu on the left.
3. Under *Identifiers*, select **Merchant IDs**, click on the Merchant ID that was setup in previous steps and then click **Edit**.
4. Scroll down to the **Apple Pay on the Web** section and click **Create Certificate** under the **Merchant Identity Certificate** section.
5. Click **Continue**, then **Choose File**, and select the Merchant Identity CSR file provided to you by Fat Zebra (ending in *-applepay-web.csr*), then click **Continue**.
6. On the **Your Certificate is Ready** screen, click **Download** to save the generated certificate.

7. Send the generated certificate to your Fat Zebra account representative. Fat Zebra will load this certificate into our system, enabling your Apple Pay integration.

Verify Your Merchant Domain

You will need to verify your Merchant Domain with Apple only if you plan to support Apple Pay on the Web. You will need to verify domains for any website that you plan to implement Apple Pay on.

1. Sign in to the Apple Developer Dashboard.
2. In *Certificates, Identifiers & Profiles*, choose **iOS, tvOS, watchOS** from the pop-up menu on the left.
3. Under *Identifiers*, select **Merchant IDs**, click on the Merchant ID that was setup in previous steps and then click **Edit**.
4. Scroll down to the **Apple Pay on the Web** section and click **Add Domain** under the **Merchant Domains** section.
5. Enter the domain name. Please note that the domain must have a valid TLS certificate in place - an error will be returned if Apple cannot verify the certificate.
6. Click **Continue**.
7. Once the domain is registered you will be provided a link to download a verification file. Download this file make it accessible at the URL specified on the screen.
8. Depending on the domain specified you may need to Verify the domain, and retry this until the domain is successfully verified. Once verification is complete the file should not be deleted, as Apple periodically checks this.

Updated over 4 years ago

← Apple Pay

Get Apple Pay Session →

Did this page help you? Yes No

Get Apple Pay Session

https://docs.fatzebra.com/docs/get-apple-pay-session

Get Apple Pay Session

Get an opaque Apple Pay session object that can be used to call the Apple Pay Javascript SDK's completeMerchantValidation method

In order to display the Apple Pay payment sheet when integrating Apple Pay on the Web, you will need to retrieve an opaque Apple Pay session object. The following Fat Zebra API endpoint can be used to retrieve the opaque Apple Pay session object.

Request Details

Property	Description
Method	`GET`
URL	**Sandbox:** https://paynow.pmnts-sandbox.io/v2/apple_pay/payment_session **Production:** https://paynow.pmnts.io/v2/apple_pay/payment_session
Authentication	Refer to Authentication

Request Parameters

Parameter	Type	Description
`url`	String	The Validation URL provided by the Apple Pay JS SDK's **onvalidatemerchant** event handler The Validation URL provided must come from one of the whitelist Apple Pay domains or a `400 Bad Request` response will be returned
`domain_name`	String	The fully qualified domain name of your website that initiated the Apple Pay payment sheet via an Apple Pay button
`display_name`	String, maximum 64 UTF-8 characters	The display name for the merchant in the Apple Pay Session. This will be displayed in the Touch Bar, and in the Payment Sheet when displayed to the customer

Example Request

Get Payment Session Request

```
curl https://paynow.pmnts-sandbox.io/v2/apple_pay/payment_session\?url\=https://apple-pay-gateway-cert.app
```

Example Response

Opaque Apple Pay Session response

```
{
  "epochTimestamp": 1597042910373,
  "expiresAt": 1597046510373,
  "merchantSessionIdentifier": "<identifier>",
  "nonce": "<nonce>",
  "merchantIdentifier": "<merchant-identifier>",
  "domainName": "fatzebra.com",
  "displayName": "Fat Zebra",
  "signature": "<signature>"
}
```

Updated almost 2 years ago

← Setup your Apple Merchant ID

Recurring & Installment Transactions →

Did this page help you? Yes No

Recurring & Installment Transactions

https://docs.fatzebra.com/docs/apple-pay-recurring-installment-transactions

Recurring & Installment Transactions

Merchants can make recurring and installment transactions using Apple Pay cards. This is achieved by using the `card_token` from the initial Apple Pay transaction where the customer authorized the recurring or installment cycle, as well as specifying the `authorization_tracking_id` returned on the initial Apple Pay transaction in all subsequent recurring or installment transactions.

> 📘 Apple have published a set of Human Interface Guidelines that detail how Apple Pay transactions should be presented to customers. These guidelines include recurring and installment payments and should be followed by merchants.

This process would look like:

1. Process an initial Apple Pay transaction

Merchants must first have the customer authorize an initial Apple Pay transaction. This can be done by following the steps in Fat Zebra's Apple Pay documentation.

2. Store the card_token & authorization_tracking_id for future use

The last step of processing the initial Apple Pay transaction is to make an API call to Fat Zebra to **create a purchase using a wallet**.

In the response of this API call you will receive two values which should be stored for later use:

- `card_token` : this is a token for the card number used in the initial Apple Pay transaction and can be used in subsequent transactions so that the actual card number does not need to be entered
- `metadata.authorization_tracking_id` : this is an ID returned by the card schemes which must be presented on subsequent recurring or installment transactions.

> ❗ Failure to present the `authorization_tracking_id` may result in the recurring or installment transaction being declined

For reference, here's an example response from the API call used to create the initial Apple Pay transaction:

Response from create a purchase with a wallet API call


```
{
    "successful": true,
    "response": {
        "authorization": "123615",
        "id": "001-P-1DAY0JLF",
        "card_number": "442769XXXXXX5460",
        "card_holder": "ApplePay Card Holder",
        "card_expiry": "2023-12-31",
        "card_token": "zce2sfg5dgryi7a3x4fm",
        "card_type": "VISA",
        "card_category": "Debit",
        "card_subcategory": "Commercial",
        "amount": 1,
        "decimal_amount": 0.01,
        "successful": true,
        "message": "Approved",
        "reference": "9dffa525-d13a-4000-ac01-80a60a39435d",
        "currency": "AUD",
        "transaction_id": "001-P-1DAY0JLF",
        "settlement_date": "2020-10-27",
        "transaction_date": "2020-10-27T09:56:05+11:00",
        "response_code": "00",
```

```
        "captured": true,
        "captured_amount": 1,
        "rrn": "812423498214",
        "cvv_match": "U",
        "metadata": {
            "authorization_tracking_id": "VI457139255729342",
            "original_transaction_reference": ""
        },
        "addendum_data": {}
    },
    "errors": [],
    "test": false
}
```

Given the above response, you would store the following values in your system for future use:

- **card_token**: `zce2sfg5dgryi7a3x4fm`
- **authorization_tracking_id**: `VI457139255729342`

3. Process recurring or installment Apple Pay transactions

To create a recurring or installment transaction, call the Create a purchase with a token API endpoint and specify a recurring or installment value in extra.ecm.

You will also need to specify the `authorization_tracking_id` value stored in step 2 above in the `extra.authorization_tracking_id` of your request:

Create a recurring Apple Pay purchase

```
curl https://gateway.pmnts-sandbox.io/v1.0/purchases -u TEST:TEST -d'
    {
        "amount": 1000,
        "reference": "oiuwiouwxmnx23",
        "customer_ip": "111.222.111.123",
        "currency": "AUD",
        "card_token": "zce2sfg5dgryi7a3x4fm",
        "extra": {
          "ecm": "32",
          "authorization_tracking_id": "VI457139255729342"
        }
    }'
```

In the above example an ecm of `32` - Internet/Recurring has been specified. This ecm value can be modified depending on your requirements, e.g. if you need a Telephone Order/Installment transaction you would specify an ecm of `13`. Refer to the following documentation for more information on ecm values.

Updated almost 4 years ago

← Get Apple Pay Session

Google Pay™ →

Did this page help you? Yes No

Google Pay™

A faster, safer way to pay. Google Pay™ allows your customers to securely and quickly check out in apps and on the web.

Google Pay Checkout Workflow

- A customer adds a credit, debit, or other payment method into Google Pay. This occurs when the customer adds payment method details into the Google Pay Android or Web app, or when the customer uses a payment method to buy a Google product or service (like an app or movie on Google Play, or storage space for Google Drive).
- The customer chooses to Buy with Google Pay on the merchant's website or Android app and only needs to then select their card and shipping address.
- The merchant receives an encrypted Google Pay token and sends this to Fat Zebra. Your systems are not exposed to the PCI Scope for this data.

Implementation Methods

There are three methods that merchants can choose from to implement Google Pay with Fat Zebra:

1. **Android App**: integrate Google Pay into an Android application.
2. **Web Integration**: integrate Google Pay into your website using the Google Pay API Javascript client library.
3. **Via Fat Zebra Hosted Payments Page**: integrate Google Pay into your website by iframing or redirecting to the Fat Zebra Hosted Payments Page. The Fat Zebra Hosted Payments Page will then handle the Google Pay checkout flow for you.

For instructions on each implementation method, click on the relevant link above.

Testing

To test Google Pay you must log in to a real Google account - to gain access to the Test Card Numbers provided by Google you must visit the following group and add yourself as a member: https://groups.google.com/g/googlepay-test-mode-stub-data

The Google Pay API will not return live, chargeable payment information in Google's test environments. The test environment can be configured on both Android App and Web Integrations - refer to Google's About the test environment documentation for more information.

The Fat Zebra Sandbox environment will accept the non-chargeable payment information returned by the Google Pay API test environment.

All Google Pay transactions sent to the Fat Zebra API Sandbox environment will return cent-based responses, whereby the response code returned will be dependent on the amount specified in the request. E.g. for a request of $1.00, the Fat Zebra API will return a response code of "00" based on the cents of the amount. Similarly a request for $1.05 will return a response code of "05" declined. This allows you to test your integration against various response codes.

Fat Zebra Environment	Google Pay Environment
Sandbox	`ENVIRONMENT_TEST`
Production	`ENVIRONMENT_PRODUCTION`

Additional Information

Supported Authorization Methods

Authorization Method
`PAN_ONLY`
`CRYPTOGRAM_3DS`

Supported Card Networks

Card Network
`AMEX`
`JCB`
`MASTERCARD`
`VISA`

Billing Address Requirements

The Fat Zebra API does not require any billing address details to be sent. This is subject to change in the future.

Limitations

> ### Limited Acquirer Support For Recurring Transactions
>
> Processing recurring transactions via Google Pay is not supported by all Australian acquirers. If these methods are used where support is not yet available the transactions may be declined or return errors.

Updated over 1 year ago

← Recurring & Installment Transactions

Android App Integration →

Did this page help you? Yes No

Android App Integration

https://docs.fatzebra.com/docs/android-app-integration

Android App Integration

Instructions detailing how to integrate Google Pay™ into an Android App using Fat Zebra as your gateway

Pre-requisites

1. Read the Google Pay Android Brand Guidelines
2. Follow Google Pay's Deploy production environment guidelines. In this step you will obtain a **merchantID**. By integrating Google Pay, you agree to Google's terms of service.
3. Contact the Fat Zebra support team to have Google Pay enabled on your account. You will be supplied a **gatewayMerchantID**.

Implementation Steps

Follow the instructions in the following Google guides to implement a Google Pay button in your app:

- Google Pay Android developer documentation
- Google Pay Android integration checklist

When setting up the *tokenizationSpecification* object, specify the following values for the **gateway** and **gatewayMerchantId** values:

Java

```
private static JSONObject getTokenizationSpecification() {
  JSONObject tokenizationSpecification = new JSONObject();
  tokenizationSpecification.put("type", "PAYMENT_GATEWAY");
  tokenizationSpecification.put(
      "parameters",
      new JSONObject()
          .put("gateway", "fatzebra")
          .put("gatewayMerchantId", "<provided to you by Fat Zebra>"));

  return tokenizationSpecification;
}
```

The Google Pay button will generate an encrypted token that you must then include in a request to the Fat Zebra API to:

- Create a purchase using the card in the Google Pay payload
- Or tokenize the card details in the Google Pay payload

The following documentation outlines the format of the two API calls:

- Create a purchase using a wallet
- Tokenize a card with wallet credentials

Updated over 4 years ago

← Google Pay™

Web Integration →

Did this page help you? Yes No

Web Integration

https://docs.fatzebra.com/docs/web-integration

Web Integration

Instructions detailing how to integrate Google Pay™ into your Website using Fat Zebra as your gateway

Pre-requisites

1. Read the Google Pay Web Brand Guidelines
2. Follow Google Pay's [Deploy production environment guidelines](Follow Google Pay's Deploy production environment guidelines). In this step you will obtain a **merchantID**. By integrating Google Pay, you agree to Google's terms of service.
3. Contact the Fat Zebra support team to have Google Pay enabled on your account. You will be supplied a **gatewayMerchantID**.

Implementation Steps

Follow the instructions in the following Google guides to implement a Google Pay button in your app:

- Google Pay Web developer documentation
- Google Pay Web integration checklist

When setting up the *tokenizationSpecification* object, specify the following values for the **gateway** and **gatewayMerchantId** values:

JavaScript

```
const tokenizationSpecification = {
  type: 'PAYMENT_GATEWAY',
  parameters: {
    'gateway': 'fatzebra',
    'gatewayMerchantId': '<provided by Fat Zebra>'
  }
};
```

The Google Pay button will generate an encrypted token that you must then include in a request to the Fat Zebra API to:

- Create a purchase using the card in the Google Pay payload
- Or tokenize the card details in the Google Pay payload

The following documentation outlines the format of the two API calls:

- Create a purchase using a wallet
- Tokenize a card with wallet credentials

Updated over 4 years ago

← Android App Integration

Hosted Payments Page Integration →

Did this page help you? Yes No

Hosted Payments Page Integration

https://docs.fatzebra.com/docs/hosted-payments-page-integration

Hosted Payments Page Integration

Instructions detailing how to integrate Google Pay™ into your Website using the Fat Zebra Hosted Payments Page

Pre-requisites

- Read the Google Pay Web Brand Guidelines
- Read the Google Pay APIs Acceptable Use Policy. You must adhere to this policy when integrating the Fat Zebra Hosted Payments Page.
- Prior to implementing Google Pay, contact the Fat Zebra support team to have Google Pay enabled on your account. You will be required to agree to the Google Pay Terms of Services in the Merchant Dashboard. The Fat Zebra support representative will provide instructions on how to do this.

Implementation Steps

Follow the instructions in the Fat Zebra Hosted Payments Page documentation. A Google Pay button will be displayed by default on the Hosted Payments Page once Google Pay has been enabled on your account.

If you chose to integrate the Hosted Payments Page via an iframe you will need to specify **allowpaymentrequest** on the **tag:**

iframe with allowpaymentrequest attribute

```
<iframe
  allowpaymentrequest
  height='800'
  src='https://paynow.pmnts.io/xxxx/invoices?iframe=1&nonce=7705a1dd&ts=1511737426&v=8f14f3f99769ce96b2488
  width='600'>
</iframe>
```

If your element uses the sandbox attribute, an allow-popups token should be set to allow display of the Google Pay payment sheet in a new window:

iframe with sandbox attribute

```
<iframe
  allowpaymentrequest
  sandbox="allow-popups"
  height='800'
  src='https://paynow.pmnts.io/xxxx/invoices?iframe=1&nonce=7705a1dd&ts=1511737426&v=8f14f3f99769ce96b2488
  width='600'>
</iframe>
```

Updated over 4 years ago

← Web Integration

Google Pay Merchant ID →

Did this page help you? Yes No

Google Pay Merchant ID

https://docs.fatzebra.com/docs/google-pay-merchant-id

Google Pay Merchant ID

All new merchant accounts setup in the Fat Zebra platform are configured to support Google Pay by default.

In order to setup Google Pay in your website or app you will require the **gatewayMerchantID** which is a unique value assigned to each merchant.

As this is automatically setup we have configured our system to create the **gatewayMerchantID** deterministically, so that partners and merchants can determine this themselves when using automated onboarding.

The following steps can be used to build the **gatewayMerchantID**:

1. Take the API token, and derive a SHA256 hash of the value as a hexadecimal digest - the result should be in lower case:

Ruby

```ruby
digest = OpenSSL::Digest::SHA256.hexdigest(token)
# IF required:
digest = digest.downcase
```

> 🚧 **Resulting Digest MUST be lowercase**
>
> Different systems may emit a hexadecimal digest in uppercase. It is important to ensure that you convert this to a lowercase string to ensure you have the correct value, as it is case sensitive.

2. Take the merchant username, and the first 16 characters of the digest, and join these with a hyphen (-):

Ruby

```ruby
merchant_id = "#{username}-#{digest[0, 16]}"
```

3. The final resulting ID will look something like the following: `username-ce91989e4d0f0fd1`

A full example (in Ruby) is below:

Ruby

```ruby
username = "TESThooli"
token = "738439923571d79914391d257afb0c3e"

digest = OpenSSL::Digest::SHA256.hexdigest(token)

merchant_id = "#{username}-#{digest[0, 16]}"
```

With these inputs, the resulting value will be `TESThooli-63b648806b4eba75` - you can use these inputs and this resulting value to test your code to ensure the values match.

Updated 3 months ago

Did this page help you? Yes No

Overview

https://docs.fatzebra.com/docs/webhooks-overview

Overview

A Webhook is an out of band communications mechanism which allows our systems to sent events to external systems, such as the notification of a payment, batch file completion, etc.

Normally webhooks are used where notifications from an automated or asynchronous process are required, or if additional systems (apart from the one interacting with the gateway) require notification of events.

Updated over 5 years ago

Google Pay Merchant ID

Handling Webhooks

Did this page help you? Yes No

Handling Webhooks

https://docs.fatzebra.com/docs/handling-webhooks

Handling Webhooks

3rd Party System Responses

When you receive webhooks the Gateway will consider the webhook successfully sent if it receives a HTTP 200 or HTTP 201 response. The system which delivers the webhooks will not follow redirects (e.g. HTTP 301 or HTTP 302), and will consider these responses as failed.

Retries

Failed webhooks will be retried on an exponential backoff, where each time a request fails, it will be pushed back onto the processing queue, and a delay will be added, growing each time until finally the retries are exhausted.

Webhooks will be attempted for up to 2 days until they are eventually considered permanently failed and will not be re-attempted.

Idempotency

As webhooks are asynchronous and are transmitted across the internet there is always a chance that the successful posting of a webhook from the Gateway into the receiving system could fail, such as due to network timeouts, receiving end errors etc.

Because of this it is recommended that the systems receiving the webhooks act on the events idempotently. This means that if you depend on webhooks to, for example, dispatch an order, it is important that you check the status of that order in your system's database before executing the workflow to dispatch the order, just in case multiple webhooks for the same subject are received.

Updated about 6 years ago

← Overview

Events →

Did this page help you? Yes No

Events

https://docs.fatzebra.com/docs/events

Events

Each webhook that is sent is the result of an event within the payment gateway. These events are usually named with the following convention:

`subject:event`

For example, when a purchase succeeds your system (if configured) may receive the `purchase:success` event.

Currently the following events are published out of the gateway:

Name	Description
purchase:success	A purchase has successfully been processed
purchase:failed	A purchase was not processed successfully
refund:success	A refund has successfully been processed
refund:failed	A refund was not processed successfully
direct_entry:submitted	A direct entry was submitted to the bank for processing
direct_entry:completed	A direct entry successfully completed processing
direct_entry:rejected	A direct entry was rejected by the bank and did not successfully complete processing
direct_entry:disbursed	A direct entry was disbursed
payment_plan:activated	A payment plan has been activated
payment_plan:suspended	A payment plan was suspended
payment_plan:cancelled	A payment plan was cancelled
payment_plan:completed	A payment plan was completed
payment_plan_payment:completed	A payment plan payment was completed
payment_plan_payment:declined	A payment plan payment was declined
payment_plan_payment:error	A payment plan payment failed due to an error
dispute:opened	A new dispute has been opened
dispute:status_changed	The status of an existing dispute has changed
card_account:update	There is an update to an existing card account - see Card Account events

Filtering Events

New webhook events may be added as the gateway evolves, so it is important that you ensure the systems receiving these events checks the event type before attempting to process the data received.

If an event is received that is not supported it is recommended that you discard the request.

Updated over 1 year ago

← Handling Webhooks

Example Webhooks →

Did this page help you? Yes No

Example Webhooks

https://docs.fatzebra.com/docs/example-webhooks

Example Webhooks

Event Structure

For each webhook that is sent to the receiving system, a JSON payload will be included with the following structure:

Name	Type	Description
`event`	string	The event name in the format of `type:event`, such as `purchase:success`
`payload`	Array or Object	The payload of the webhook. Usually this will either be the record as a hash you would normally receive via the API, or an array of hashes, where a webhook is triggered at one time for multiple objects

For Webhooks triggered as a result of batch operations (for example, direct entry completion or rejection) the request sent will contain an array of objects under the `payload` field. Please ensure that your code checks for an object or array before processing the request.

Purchases

purchase:success | purchase:failed


```
{
  "event": "purchase:success",
  "payload": {
    "authorization": "389768",
    "id": "071-P-245JAGI0",
    "card_number": "400555XXXXXX0001",
    "card_holder": "Joe Bloggs",
    "card_expiry": "2023-05-31",
    "card_token": "fke8tr3e",
    "card_type": "VISA",
    "card_category": "Credit",
    "card_subcategory": "Standard",
    "amount": 123,
    "decimal_amount": 1.23,
    "successful": true,
    "message": "Approved",
    "reference": "0k7uchjosk38",
    "currency": "AUD",
    "transaction_id": "071-P-245JAGI0",
    "settlement_date": "2018-09-07",
    "transaction_date": "2018-09-07T17:06:00+10:00",
    "response_code": "00",
    "captured": true,
    "captured_amount": 123,
    "rrn": "071P245JAGI0",
    "cvv_match": "M",
    "metadata": {

    },
    "addendum_data": {

    }
  }
}
```


Refunds

refund:success | refund:failed

```
{
  "event": "refund:success",
  "payload": {
    "authorization": "1533336051",
    "id": "071-R-3RCHNEJT",
    "amount": 1000,
    "refunded": "Approved",
    "message": "Approved",
    "card_holder": "Jim Smith",
    "card_number": "512345XXXXXX2346",
    "card_expiry": "2023-05-31",
    "card_type": "MasterCard",
    "transaction_id": "071-R-3RCHNEJT",
    "reference": "rmt1533336049913",
    "currency": "AUD",
    "successful": true,
    "transaction_date": "2018-08-04T08:40:51+10:00",
    "response_code": "00",
    "settlement_date": "2018-08-05",
    "metadata": {

    },
    "standalone": false,
    "rrn": "071R3RCHNEJT"
  }
}
```


Direct Entries

direct_entry:created | direct_entry:submitted | direct_entry:completed | direct_entry:rejected


```
{
  "event": "direct_entry:created",
  "payload": {
    "id": "1229-DD-C3VM4MXV",
    "amount": 10.0,
    "bsb": "085-124",
    "account_number": "091765432",
    "account_name": "Bob Smith",
    "description": "DD INV002",
    "reference": "DD1357924680004",
    "date": "2019-12-25",
    "process_date": null,
    "status": "New",
    "result": null,
    "type": "Debit",
    "metadata": {}
  }
}
```


Payment Plans

payment_plan:active | payment_plan:completed | payment_plan:suspended | payment_plan:cancelled


```
{
  {
    "event": "payment_plan:active",
    "payload": {
      "id": "1486-PP-ECY8AH07",
      "customer": "1486-C-9R9FY0FE",
      "amount": 25000,
      "currency": "AUD",
      "setup_fee": 0,
      "frequency": "Monthly",
      "anniversary": 1,
      "start_date": "2018-08-04",
      "end_date": null,
      "total_count": null,
      "total_amount": null,
      "payment_method": "Credit Card",
      "reference": "Plan456",
      "description": "",
      "status": "Active",
      "status_reason": "None",
      "created_at": "2018-08-03T10:28:09.901+10:00",
      "failed_payment_fee": 0,
      "retry_interval": 3,
```

```
      "status_change_date": null,
      "payments": [
        {
          "id": "1486-PT-ASASNHVE",
          "payment_plan": "1486-PP-ECY8AH07",
          "reference": "Plan456-0001",
          "amount": 25000,
          "currency": "AUD",
          "scheduled_date": "2018-09-01",
          "payment_method": "Credit Card",
          "status": "Scheduled",
          "result": null,
          "records": []
        }
      ]
    }
  }
}
```


Payment Plan Payments

payment_plan_payment:completed | payment_plan_payment:declined | payment_plan_payment:error


```
{
  {
    "event": "payment_plan_payment:completed",
    "payload": {
      "id": "787-PT-G1EBHXZR",
      "payment_plan": "787-PP-4745AS4X",
      "reference": "PPAPI-001-0400",
      "amount": 10,
      "currency": "AUD",
      "scheduled_date": "2049-04-23",
      "payment_method": "Credit Card",
      "status": "Completed",
      "result": "Ad-Hoc Payment",
      "records": [
        {
          "authorization": "389768",
          "id": "071-P-245JAGI0",
          "card_number": "400555XXXXXX0001",
          "card_holder": "Joe Bloggs",
          "card_expiry": "2023-05-31",
          "card_token": "fke8tr3e",
          "card_type": "VISA",
          "card_category": "Credit",
          "card_subcategory": "Standard",
          "amount": 123,
          "decimal_amount": 1.23,
          "successful": true,
          "message": "Approved",
          "reference": "PPAPI-001-0400",
          "currency": "AUD",
          "transaction_id": "071-P-245JAGI0",
          "settlement_date": "2018-09-07",
          "transaction_date": "2018-09-07T17:06:00+10:00",
          "response_code": "00",
          "captured": true,
          "captured_amount": 123,
          "rrn": "071P245JAGI0",
          "cvv_match": "M",
          "metadata": {},
          "addendum_data": {}
        }
      ]
    }
  }
}
```


Disputes

dispute:opened | dispute:status_changed


```
{
  "event": "dispute:opened",
  "payload": {
    "id": "001-P-3V8QW1BR",
    "reference": "MERCHREF-001",
    "dispute_reference": "ABC-123939847589437123412342320221205",
```

```
    "status": "open",
    "opened_on": "2022-12-06",
    "reason_code": "4387",
    "reason_description": "Card Not Present (fraud)",
    "amount_cents": 10000,
    "amount_currency": "AUD",
    "next_due_date": "2022-12-20"
  }
}
```

Card Account

card_account:update

```
{
  "event": "card_account:update",
  "payload": {
    "token": "abcd1234abcd",
    "card_number": "555555XXXXXX1243",
    "card_expiry": "2021-09-30",
    "card_type": "MasterCard",
    "card_category": "Credit",
    "card_updater_status": "active"
  }
}
```

Updated over 1 year ago

← Events

Card Account events →

Did this page help you? Yes No

Card Account events

https://docs.fatzebra.com/docs/card-account-events

Card Account events

For merchants taking advantage of network tokens, this webhook will fire when:

- a change has occurred to a card-on-file. For example, when a card's expiry date has been updated, merchants will receive an update event with the new expiry date;
- the scheme token status is updated.

The network token status will be one of the following values:

Status	Description
pending	This card has been enrolled for network tokenisation but has not yet had its token activated. Any transactions attempted with a card in this state will fall back to the original PAN.
active	This card has an active network token. When transacting with a card in this state, a digital PAN and corresponding cryptographic signature will be fetched and used for the transaction. If there are errors when fetching the payment instrument (for instance due to service interruptions), the transaction will fall back to the original PAN.
suspended	This card has a network token, however the token has been suspended by either the network or issuing bank. A card that is suspended may return to the active state, or it may be deleted. Any transactions attempted with a card in this state will fall back to the original PAN.
deleted	This card has had its network token deleted, possibly due to account closure or some other reason. Merchants should attempt to contact the card holder for an updated payment method. Any transactions attempted with a card in this state will fall back to the original PAN.

card_account:update


```
{
  "event": "card_account:update",
  "payload": {
    "token": "abcd1234abcd",
    "card_number": "555555XXXXXX1243",
    "card_expiry": "2021-09-30",
    "card_type": "MasterCard",
    "card_category": "Credit",
    "card_updater_status": "active"
  }
}
```


Updated over 1 year ago

← Example Webhooks

Bring Your Own Network Token →

Did this page help you? Yes No

Bring Your Own Network Token

https://docs.fatzebra.com/docs/byot

Bring Your Own Network Token

FatZebra allows the use of network tokens for Mastercard or Visa through the MDES and VTS program.

Merchants may wish to perform network tokenisation of their own cards to manage lifecycle events of these cards. FatZebra allows the presentment of network tokens with the cryptogram, passing these through to the switch.

By supplying a network token as the card number, the token expiry as the card expiry and the network provided cryptogram in the cryptogram field, FatZebra will mark the card as being a provided network token.

All other fields are applicable as for a regular purchase.

json

```
{
  "card_number": "5555555555555555",
  "card_holder": "Valued Cardholder",
  "card_expiry": "02/2027",
  "customer_ip": "127.0.0.1",
  "amount": 110,
  "currency": "AUD"
  "cryptogram": "Aaaaaa11111222222333334444==="
}
```

All subsequent transactions, either using the network token as the card number or the returned FatZebra token card will be treated as a network token transaction.

The cryptogram should be fetched as per rules for the network token - through either VTS or MDES.

On subsequent transactions, if the cryptogram is not present, it will be omitted. This is **not** recommended; some transactions are eligible to have the cryptogram omitted - for instance merchant initiated transactions that are recurring - however they may be declined by the issuer.

Updated 2 months ago

← Card Account events

Overview →

Did this page help you? Yes No

Overview

https://docs.fatzebra.com/docs/overview-1

Overview

Security is our highest priority - each year we go through an audit process which covers a comprehensive set of guidelines provided by the PCI Security Standards Council.

In addition to our yearly audit we scan our infrastructure for any vulnerabilities (on our servers) and an external scanning vendor checks for any problems in our website, dashboard and gateway. All of this is part of what we do to ensure card holder data stays secure.

Updated over 4 years ago

← Bring Your Own Network Token

PCI Certification →

Did this page help you? Yes No

PCI Certification

https://docs.fatzebra.com/docs/pci-certification

PCI Certification

Fat Zebra is a PCI DSS Tier 1 certified service provider. Essentially this means that we have passed a yearly audit from a Qualified Security Assessor (QSA) – in our case, PCI Consulting Australia. This audit covers 12 different requirements including physical security, the storage and transmission of credit card data, data security and more.

For mode details on the PCI-DSS requirements please visit the PCI-DSS website.

Encryption

In order to protect your customers data, your passwords and your information we encrypt information stored within our database and information transmitted between website users and Fat Zebra's website.

Passwords are stored as a non-reversible hash (this means that if our database is compromised your passwords will not be leaked, and are required to be changed every 90 days. We recommend that you use a passphrase, instead of a password - this is commonly 4 or 5 words which you can remember easily but is still secure.

Credit Card data is encrypted with asymmetric encryption, with a keypair uniquely assigned to your merchant account, along with a 2048-bit passphrase required to decypt the card data. Card data decryption is a bit more complicated - in order to unlock the private key associated with your merchant account an approved operator must provide their own secure passphrase. This allows for quick revocation of keys and rotation of encryption keys regularly.

At the end of the day this may seem complicated, but in testing and in practice we have found this is secure and fast. With security being one of our biggest concerns we're happy to have things a little bit more complicated if it means peace of mind.

Physical Security

Fat Zebra hosts its equipment, including its Cardholder Data Environment (CDE) within a secure data centre located in Canberra. Access to the computer room facilities require the following:

- Identification is presented and mobile phones/cameras are surrendered
- Visitors sign in and passes are issues
- If after hours, the computer room master key is obtained from a Class B security safe (each visitor has individually assigned access codes)
- The Class 3 alarm is disarmed with the visitors unique access code and the swipe pass is used to access the computer room doors.
- The visitors sign into the computer room
- The key to the secured rack is obtained from the key safe, which is accessed with the swipe pass
- Signing out (and if necessary, re-arming and locking of the computer room) is required before surrendering passes and retrieving identification and mobile phones
- In addition to this, access to our hosting racks is restricted to authorized Fat Zebra staff, however data centre operations staff are granted access with permission from Fat Zebra.

Data Security and Redundancy

To ensure that your data is always available we have redundant systems in place including a highly available database cluster with off site backups, redundant power, backup power supplies from diesel generators (provided within the data center), redundant cooling and redundant network links.

Intrusion Detection

Fat Zebra have implemented an industry standard Intrusion Detection System which monitors all traffic on our network for any potentially malicious traffic. When this system detects malicious traffic our security staff are notified, who then investigate the event to determine whether or not further action is required.

Disclosure

We believe that as a payment gateway, trust between us and our merchants is the most important aspect of doing business.

In the event of any security vulnerabilities, intrusion attempts, data theft/loss or successful intrusions we commit to inform our merchants, merchant banks and the PCI-DSS council as soon as we are aware of the issues. If you have any questions please feel free to contact us.

Accountability

Fat Zebra staff should never need to see your merchant account, unmasked credit card numbers or your transaction data, except for the following situations:

- Troubleshooting and support
- Security Investigations
- Requests from regulation or law enforcement bodies
- Exceptional situations deemed necessary by the directors of Fat Zebra

If credit card numbers need to be unmasked by authorized Fat Zebra staff appropriate procedures must be followed, and all decryption operations are recorded with full justification.

Updated over 4 years ago

WHAT'S NEXT

If required you can request a copy of our Attestation of Compliance to provide to assessors and retain on file for compliance requirement.

Request PCI Documents

Did this page help you? Yes No

What is PCI Compliance

https://docs.fatzebra.com/docs/what-is-pci-compliance

What is PCI Compliance

The Payment Card Industry Data Security Standard (PCI DSS) is a set of requirements designed to ensure that ALL companies that process, store or transmit credit card information maintain a secure environment. Essentially any merchant that has a Merchant ID (MID).

Motivation to become compliant? The incentives include a 'Safe Harbour' from certain penalties and fines if a merchant is compliant at the time of breach.

Who needs to be compliant?

PCI Compliance is required for any company which sees, processes, holds or handles debit or credit card details in an electronic form, which could apply to your website, your retail shop and/or your office. In practice, this means any company which has a merchant bank account.

This applies if you take payment by card in your shop, or over the phone, or using a virtual terminal, or on your website using Fat Zebra. If you take payment on your website using Google Checkout, then the answer is no, as you don't have a real merchant bank account.

Where can I find out about the PCI Data Security Standards

The standards can be found on the PCI DSS website.

Do organizations using third-party processors have to be PCI compliant?

Yes. Merely using a third-party company does not exclude a company from PCI compliance. It may cut down on their risk exposure and consequently reduce the effort to validate compliance. However, it does not mean they can ignore PCI.

How do I become compliant?

PCI Compliance has two parts: a questionnaire about how you protect the card data you handle, and sometimes a software scan on your website, office and/or shop to make sure it is secure. The questionnaire and scan basically ensure you meet the list of PCI requirements, which include things like having a secure network, regular testing and a security policy.

What will happen if I don't bother?

If you have an Internet Merchant Facility, your bank may eventually start fining you until you are compliant, or put a freeze on your account until you become compliant.

Should your website or application be hacked, and card details are stolen, you may be liable for crippling fines and asked to conduct a full PCI Audit by a QSA before your bank allows you to start taking payments again.

Updated 12 months ago

← PCI Certification

PGP Key →

Did this page help you? Yes No

PGP Key

https://docs.fatzebra.com/docs/pgp-key

PGP Key

Use our PGP key to encrypt and/or sign your sensitive content - we have provided details of our PGP key below.

Once you have encrypted your content with our PGP key please email is through to security@fatzebra.com, or if the content is too large reach our to our Support Team for details on how to send your file.

If you have any questions, or run into any problems, please get in touch.

PGP Key Details

View this key on the OpenPGP Key Server

- Key ID: 2EA4E3E2
- Key Type: RSA
- Key Size: 4096
- Fingerprint: 9A84 023B 95E2 E93F E220 1347 70BD 2980 2EA4 E3E2
- User ID: security@fatzebra.com.au

Text

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBFUNbk8BEADBcnVIjNPlKSZqknaqQqZwWdmTgS+2QvhaKclfTzXpbo2weio5
vVvNnYeMzP+qZayKtMR+1zfDwNSvmmzo2u6EXXS+xzslilh8oXhnWGFl2gtebcNH
a0+huY1NLNV2N2IVYENBqToyyVrbewU8PIYdZ7JqxLRcY/egKTUx7vL9DXhRA9ce
kuz4EKySv2goEiOe9Y6YiIXbA5nOuBbc9hJ+xWsdrG1LfgxwXjyFnVk4i5ewlO5v
awCggtZP/7R3DQVH8dxxtjZ4GUZhD1YpjrxESoVe3Z87SrBLBvldacHYcOF+MqI3
H6wIDmBvftPz5pLAaObHa+uWlR6Vp0slvwIBySTvfUGNDpbhZ3pj7S4Gzt5SXy9y
kHF2+4nN0IXk8850ONwtL2SbLJnosrmJpN4SIJOaxcwKkoE8EKVAawF2F/fjbfuk
CekjaEPYEv9la01bVdOAoddq5rVc+ECnI0YyMBWSrjkb+QLHvMZZR3hXAk+T5CQ1
+wvnxaPFWtuj+/WbsHiuq9PEDOIhH8wLGR6jFSa223xF0Ce0uEEFjzbQzyIUjFqA
kfAmEI76iC7bvsqy0ovr/8DFT+t5v7mkM/+IFsNtIOqpzsLHhKcZ+iE+sW9JaKsV
OP2RrR4WqTmd0Io9VMW5yvTtkmD+pXJeVzC/82X6rys+JcxlvT6H0X60oQARAQAB
tCRGYXQgWmVicmEgPHNlY3VyaXR5QGZhdHplYnJhLmNvbS5hdT6JAlQEEwEKAD4C
GwMFCwkIBwMFFQoJCAsFFgIDAQACHgECF4AWIQSahAI7leLpP+IgE0dwvSmALqTj
4gUCZGGn7gUJFqT0lQAKCRBwvSmALqTj4rySD/0T1Utoq8RXniTXrB5eII0euRDx
bzpKgGfvsBJ9+O/xFu/ovWeO8nP9nt/fm59F1xTbmGycY+2EBIPOlV9VJCWff8Ee
8yaFf00i1WdIaaBGLlwo2TStQHNNS8xgrn0BsgbpbDTtV50hC7SAqafaEynvH3J1
UyqvL0fF61B8Hr7NfEffSGXkMAWquWoBBFR4wGfrUQdE6ehGb4NbWF6ePZWpiJjN
BdWf9T2jL/NlmordsvMTS9zpP7UQ3mu86jxBuYjtfIc8rrgP9b63fnx+SVFWIqtb
oRsVfEuJYVy+EkqUHt+sVBTQ3ls0JbjEbOWwylIr9GzWuMHM1CqvvRhCT4Mt9+9B
QUNmQ+4QBfVaxTc2mrSpQGsS/zTuPfzuFIASYKjk0IG7XcQm5v0G/3RvbDksSTVI
6LHGzuju4FbNtB5JGH7cTd2t1uM4io8DlLcKQ4hS4J33mgx1oAiEGQ6PYgpp1oFn
pmF2efOYrurAzvObSFSHQempFPulCmEf1xUu3PToVtniOK1na/qWpBIHd03L69vP
42gfMjpKR0golm8fma5RuXJaYuGGu1ijgLXnz9Atnb4J+zVdI+pe15+x7aiiHfl6
H8RfKKkPiJvvQuKM8smUmibNBBQUnlZ6hysXR4Py2qHSVQ0fyQxKfhQEmj5nbKzk
Ld68Z6P7e2TAF+S7fLkCDQRVDW5PARAAvzKlT4EA1OZVtP98ohzvfzVWGVQz4CiX
BI42El8GXMUh0ZhEC1zvOgQEsz44hEkQelmyk6z1jD0+bQZZG0/4nciD15HPUWds
drytWL0d9crImoHq6fwUgGbMTRt3fYTSiJHY1mJI9xUWbr3GNZrKwmbGj6wz7qCJ
Vn+rLaKhvZMNFWkBzVpqc3VV1f8KtZFYo5MIEq+zLEn+qg4xQq/mFC9McpPcUPm/
IwxWmQOsjTWRykmqHh583g7Ken0cIodrTuP36TqJcjA4tTbuRdhiRxzQ3iLNTBsA
sNnjRD7mFvk+U4jCUDwLAhv4lZy2a36692fmLWCY9q7r4nIl5OVbXhqYvL5ZkzXh
jCrim5xbokELUREx9LWgxkJ8Zvr+52MHfUL0KX+PJoAshxmH/Bq7gKG/AIHsE9CL
labixaFJmJbwlmFKB7qLOx8Vh3vQgYaHZUkZIbuMFxNC2YsYkyzTmkrMwCCh73hF
BRWqG8Bi4A7sP1qAPbPZnCrXDLLvMryBuzLb22ai51jQR8TCzqc7lNSb6K89ePRT
1XjM5EAm/klPf8W2iiI31KvD60VnPHz9SzaIJqk6ea80n/Dnh4NEi/nJZkYmVPcN
JBaKtxm8F+Ww1I140vFicVF0ZPUqHXGTBSzYUXmEe1cnvCJoUJctVb9YeOF4LT98
UhDaK7IF0nUAEQEAAYkCPAQYAQoAJgIbDBYhBJqEAjuV4uk/4iATR3C9KYAupOPi
BQJkYaetBQkW45NYAAoJEHC9KYAupOPitFMQAK8vPs7ekRgBLbDbvRpbVvuKGFHG
qcu7gF6ZJvtHOcNwAOh+kYNlrsJeQtL6rtm4q3B/OJBSZYIBRxCR4xjIfFG/EeUu
PqMfs69O64hkVnJ/Q98NjR2HAXq9AV+BCMKUn3s+0c0np81ir/vmdRP+HGXWsokn
34Od6X4grWvfB2gUXKQddFKb/6oUgphF1R0afHxcLNWKgNxOyic7o2SntIHtJQdi
r8/HITMnnpzB8tTGwgjTYeAHDKSAx2RlwQYph2nj/dGIFD8x3PxXgfebRdUHzRQh
```

```
vCcR/bPRcbZ/ew/TT6/30BwyJuBfSKODClHA/46/kv6l2JUt+7eGllA8oZCsAWfd
TgST2xL4Lz39+NZOuSgYYYTYv5c941rSqZrxaoxeJ3V6f68Zl6LgQFlj7y1+kHhh
bQCyLo9fdXYGzV0hMgjLSIytVDcMZlDonYewRUtJBEQY46GaSi5FX/2IPWaifI4g
QvbvX2OX+/TJe3JodfPQI83JPiFS37U2tkksL3t3pFfSd/itPPGe/+ga/MUfhfjI
uW4X8ASHxW+tayVZIAQd4h/cgnseMQ3H1wuaOBwd6+itCaGDdFnR1OLqR+2Fo1cD
wJQHOi51YSr6G1S4SHkhjv+836meZWagDOHuNByqH2MhADB2oHk0HqfHNCDjPJr0
S701U7X73o88at8B
=htwZ
-----END PGP PUBLIC KEY BLOCK-----
```

Updated over 1 year ago

← What is PCI Compliance

React SDK - Overview →

Did this page help you? Yes No

React SDK - Overview

https://docs.fatzebra.com/docs/react-sdk-overview

React SDK - Overview

Introduction

Fat Zebra provides a JavaScript package called named **@fat-zebra/sdk** that enables merchants to access features including:

- 3DS2
- Card tokenization

Authentication

Each feature accessible via fatzebra.js requires OAuth authentication whereby merchants obtain an OAuth access token for each payment session required. Refer to Obtain an OAuth token for more details.

Usage

First, install this package by adding it into your `package.json` as a dependency.

```
"dependencies": {
   "@fat-zebra/sdk": "^1.5"
}
```

Verify a new card

Then to use this package, we can import the relevant parts we need from the SDK. To then verify a card, we can use the `VerifyCard` component.

From this action of this component, we can pull out the token from the card tokenization event, and (if required) wait until the `PublicEvent.SCA_SUCCESS` event has happened before continuing with a customer's purchase.

```
import React from "react";
import { HmacMD5 } from "crypto-js";
import {
  Environment,
  Payment,
  PaymentIntent,
  PublicEvent,
  PaymentConfig,
  Handlers,
} from "@fat-zebra/sdk/dist";
import { VerifyCard } from "@fat-zebra/sdk/dist/react";
import { useState } from "react";

// This should be generated from your backend via your OAuth credentials.
const accessToken =
  "<your oauth access token goes here>";

export default function Checkout({ reference }: { reference: string }) {
  const [events, setEvents] = useState<Array<PublicEvent>>([]);

  const addEvent = (event: any) => {
    setEvents((prev: Array<PublicEvent>) => [...prev, event]);
  };

  const payment: Payment = {
    reference: reference,
    amount: 100,
    currency: "AUD",
  };

  // NEVER expose shared secret to client side. This is just to illustrate working example:
  const [verification] = useState(
```

```
    HmacMD5(
      `${payment.reference}:${payment.amount}:${payment.currency}`,
      "<your shared secret goes here>" // Your "Shared Secret"
    ).toString()
  );

  // These are typically sourced from your backend
  const paymentIntent: PaymentIntent = {
    payment,
    verification,
  };

  const config: PaymentConfig = {
    username: "your-username-here",
    environment: Environment.sandbox, // or Environment.production
    accessToken: accessToken,
    paymentIntent: paymentIntent,
    options: {
      sca_enabled: true,
    },
  };

  const tokenizationSuccessful = (event: any) => {
    addEvent(event);
  };

  const scaSuccess = (event: any) => {
    addEvent(event);
  };

  const scaError = (event: any) => {
    addEvent(event);
  };

  // Subscribe to the particular events you wish to handle
  const handlers: Handlers = {
    [PublicEvent.FORM_VALIDATION_ERROR]: addEvent,
    [PublicEvent.FORM_VALIDATION_SUCCESS]: addEvent,
    [PublicEvent.TOKENIZATION_SUCCESS]: tokenizationSuccessful,
    [PublicEvent.SCA_SUCCESS]: scaSuccess,
    [PublicEvent.SCA_ERROR]: scaError,
  };
```

Verify Existing Card

To then verify a card, we can use the `VerifyExistingCard` component.

From the action of this component, the successful or failed tokenised credit card event will fire. Indicating whether or not a card has been previously tokenised.

This component will return either: “Could not find card on file” or “fz.tokenization.success” status and run sca on each verification.

Calculation of verification hash

The verification hash for verify an existing card is calculated by hashing the card token. This is to be done server-side as an example:

JavaScript

```
const CARD_TOKEN = "fdsde3yx"
const SHARED_SECRET = "TEST" -- never expose this to the client
const HIDE_CARD_HOLDER = true
const hash = CryptoJS.HmacMD5(
  CARD_TOKEN,
  SHARED_SECRET
).toString()
```

React

```
import React, {useState} from "react";
import {
  Environment,
  PublicEvent
} from "@fat-zebra/sdk/dist";
import {VerifyExistingCard} from "@fat-zebra/sdk/dist/react";

// app.jsx

export default function Main() {
```

```
  const REFERENCE = "1234"
  const ACCESS_TOKEN = "< oauth access token goes here >"
  const VERIFICATION = "<this hash is calculated server side as seen above using shared secret>"

  return  <VerifyOurCard verification={VERIFICATION} accessToken={ACCESS_TOKEN} reference={REFERENCE} />

}

// VerifyOurCard.jsx
export default function VerifyOurCard(verification, accessToken, reference) {
  const FZ_CARD_TOKEN = "1234"
  const payment = {
    reference,
    amount: 100,
    currency: "AUD"
  };

  const paymentIntent = {
    payment,
    verification,
  };

  const config = {
    username: "TEST",
    environment: Environment.development, // or Environment.production
    accessToken,
    paymentIntent: paymentIntent,
    options: {
      iframe: true
    },
  };

  const [events, setEvents] = useState([]);

  const handleEvent = (event) => {
    setEvents((prev) => [...prev, event.type]);
  }

  // Subscribe to the particular events you wish to handle
  const handlers = {
    [PublicEvent.FORM_VALIDATION_ERROR]: handleEvent,
    [PublicEvent.FORM_VALIDATION_SUCCESS]: handleEvent,
  };

  return <>
    <VerifyExistingCard
      config={config}
      handlers={handlers}
      cardToken={FZ_CARD_TOKEN}
      iframeProps={{width: "100%", height: "700px"}}
    />
    <div className="flex flex-col overflow-hidden h-max-[100px]">
      {
        events.map((event, index) => <div
          style={{background: event.includes('error') ? "red" : "green"}}>
          <strong>{index + 1} {event}</strong></div>)
      }
    </div>
  </>
```

Creating a payment

There's currently no form support for taking a payment with this SDK.

Instead, we would recommend passing the token from a `TOKENIZATION_SUCCESS` event back to your backend, and then processing a payment from there using the FatZebra API.

Updated 5 months ago

← PGP Key

Obtain an OAuth token →

Did this page help you? Yes No

Obtain an OAuth token

https://docs.fatzebra.com/docs/obtain-oauth-token

Obtain an OAuth token

An OAuth access token is required to access most fatzebra.js features. To obtain an OAuth access token:

OAuth Client

1. Log into the Fat Zebra Merchant Dashboard.
2. Navigate to **Tools** > **OAuth Clients** > **Create new OAuth Client**.
3. Download your OAuth Client credentials, access key and access secret. Make sure you save them securely. For security purposes, downloading these credentials is a one-off process and you will not be able to retrieve them again.

OAuth Access Token

An OAuth access token is a signed JWT that grants fatzebra.js access to the Fat Zebra platform. It contains basic info about the merchant and a list of access grants that determine what the features the OAuth token has access to.

The token needs to be stored in browser's local storage with key space `fz-access-token` prior to initialising fatzebra.js.

If the access token is either missing or invalid, fatzebra.js will not be permitted to proceed the intended action (e.g. initiate a 3DS2 check).

For security purposes, the OAuth access token is only valid for 15 mins. It is recommended that a unique access token is created for each payment session you require.

Obtaining an access token

To obtain an OAuth access token, make a **POST** call to the OAuth endpoints listed below using your `access_key` and `access_secret` that you downloaded from the Fat Zebra Merchant Dashboard.

Environment	Endpoint
Staging	`POST` https://api.pmnts-staging.io/oauth/token
Sandbox	`POST` https://api.pmnts-sandbox.io/oauth/token
Production	`POST` https://api.pmnts.io/oauth/token

Request

Request Body

```
{
    "access_key": 'xxx',
    "access_secret": 'xxx'
}
```

Response

Successful Response

```
{
    "message": "created",
    "data": {
        "token": "xxx"
    }
{
```

Updated 12 months ago

WHAT'S NEXT

SDK Integration Sample →

Did this page help you? Yes No

Javascript SDK Features

https://docs.fatzebra.com/docs/javascript-sdk-features

Javascript SDK Features

3DS2

- Overview
- 3DS2 for Hosted Payments Page
- 3DS2 for Custom Payments Form

Updated 12 months ago

WHAT'S NEXT

Methods →

Did this page help you? Yes No

Methods

https://docs.fatzebra.com/docs/methods

Methods

fatzebra.js Methods

Updated about 4 years ago

WHAT'S NEXT

Objects →

Did this page help you? Yes No

constructor

https://docs.fatzebra.com/docs/constructor

constructor

Constructor(params)

params

Attribute	Type	Mandatory	Description
`username`	string	Y	Merchant username
`test`	string	N	Required to be `true` in the Sandbox environment. Defaults to `false` .

usage

JSON

```
const fz = new FatZebra({
  username: "MerchantXYZ",
  test: true
});
```

Updated about 4 years ago

← Methods

purchase →

Did this page help you? Yes No

purchase

https://docs.fatzebra.com/docs/purchase

purchase

If merchant decides to provide their own custom checkout button (hideButton: **true**) to work with the Hosted Payment Page loaded by **fatzebra.js**, they will need to call the *checkout* method in order to trigger the payment flow. This method sends a trigger to the Hosted Payment Page within iframe's content view.

Example

```
const fz = new FatZebra({
  username: 'xxx',
  test: true
});

fz.renderPaymentsPage({...}) // omitted

const handler = function() {
  fz.checkout();
}

document.getElementById('checkoutButton').addEventListener('click', handler);
```

Updated almost 2 years ago

← constructor

verifyCard →

Did this page help you? Yes No

verifyCard

https://docs.fatzebra.com/docs/verifycard

verifyCard

verifyCard(VerifyCardParams)

Starts the 3DS2/SCA process. This method is used for API and Direct Post integration modes only. If you are integrating 3DS2 using the Hosted Payments Page you do not need to call `verifyCard` as the Hosted Payments Page will handle this for you.

The `verifyCard` method can be called with a new card (using the raw card number) or with an existing card (using a Fat Zebra card token). If using an existing card (Fat Zebra card token), specify `card_on_file` as the `paymentMethod`. Refer to the PaymentMethod object for more details.

See VerifyCardParams for more details on the available parameters.

Usage

JSON

```
const fz = new FatZebra({
  username: "MerchantXYZ"
});

fz.verifyCard({
   customer: {
    firstName: 'Captain',
    lastName: 'America',
    email: 'hello.world@fatzebra.com.au',
    address: '123 Australia Blvd.',
    city: 'Sydney',
    postcode: '2000',
    state: 'NSW',
    country: 'AU'
  },
  paymentIntent: {
    payment: {
      amount: 1000,
      currency: "AUD",
      reference: "ref_123490",
    },
    verification: "ver_123480"
  },
  paymentMethod: {
    type: "card",
    data: {
      number: "4111111111111111",
      holder: "John Doe",
      expiryMonth: "01",
      expiryYear: "2022",
      cvv: "123"
    }
  }
});
```

Updated over 3 years ago

← purchase

renderPaymentsPage →

Did this page help you? Yes No

renderPaymentsPage

https://docs.fatzebra.com/docs/renderpaymentspage

renderPaymentsPage

renderPaymentsPage(HppSetupParams)

Renders the Fat Zebra Hosted Payments Page within the specified div element (see `containerId` in the example below).

JavaScript

```
const fz = new FatZebra({
  username: "MerchantXYZ"
});

fz.renderPaymentsPage({
  containerId: "fz-paynow",
  customer: {
    firstName: 'Captain',
    lastName: 'America',
    email: 'hello.world@fatzebra.com.au',
    address: '123 Australia Blvd.',
    city: 'Sydney',
    postcode: '2000',
    state: 'NSW',
    country: 'AU'
  },
  paymentIntent: {
    payment: {
      amount: 500,
      currency: "AUD",
      reference: "ref_123490"
    },
    verification: 'ver_123480'
  },
  options: { // Hpp display options
    hideButton: false,
    hideLogos: true
  }
});
```

Updated over 3 years ago

← verifyCard

Objects →

Did this page help you? Yes No

Objects

https://docs.fatzebra.com/docs/objects

Objects

fatzebra.js Objects

- Customer
- PaymentIntent
- PaymentMethod
- HppSetupParams
- VerifyCardParams

Updated about 4 years ago

WHAT'S NEXT

Events →

Did this page help you? Yes No

Customer

https://docs.fatzebra.com/docs/customer

Customer

The **Customer** object holds details about a customer and is used in the following methods:

- renderPaymentsPage
- verifyCard

Attribute	Type	Required	Description
`firstName`	string	Y	Customer first name
`lastName`	string	Y	Customer surname
`email`	string	Y	Customer email address
`address`	string	Y	Customer address e.g. `315/123, Success Blvd`
`city`	string	Y	Customer suburb/city
`postcode`	string	Y	Customer postcode
`state`	string	Y	Customer state
`country`	string	Y	Customer country. ISO Alpha 2 country codes. e.g. AU, GB, NZ

JSON

```
{
    firstName: 'Captain',
    lastName: 'America',
    email: 'hello.world@fatzebra.com.au',
    address: '123 Australia Blvd.',
    city: 'Sydney',
    postcode: '2000',
    state: 'NSW',
    country: 'AU'
  }
```

Updated over 3 years ago

← Objects

PaymentIntent →

Did this page help you? Yes No

PaymentIntent

https://docs.fatzebra.com/docs/paymentintent

PaymentIntent

The **PaymentIntent** object contains payment data. To ensure that this payment data is not tampered with (e.g. a customer amends the amount to be smaller), the `PaymentIntent` requires a HMAC `verification` field to be conducted in your backend using your Pay Now secret.

`PaymentIntent` is used in the following methods.

- load
- verifyCard

Attribute	Type	Description
`payment.amount`	number	Amount in subunit (non-decimal)
`payment.currency`	string	ISO-4217 country codes
`payment.reference`	string	Reference or invoice number for the payment transaction/
`payment.hide_card_holder`	boolean	Determines if the "Card Holder" field should be shown on the page or not. **If this is included and true, it must be appended to the end of the verification hash**. See "Verification Value Hash Calculation" below for more details.
`verification`	string	For payments: Hash of amount, reference, currency. For verifyCard verifyCard: Hash of card_token See **Verification Value Hash Calculation** below for more details on how to calculate this value.

Example

```
{
  payment: {
    amount: 10025,
    currency: "AUD",
    reference: "INV1121"
  },
  verification: "xxxxxx"
}
```

Verification Value Hash Calculation

> ! The Verification Value Hash should only ever be calculated on your backend server, and never in client-side code.
>
> This is because your Pay Now token is a secret value that only you should know. Calculating the Verification Value Hash on the client-side (e.g. in client-side Javascript) would make your Pay Now token secret visible to public via browser developer tools.

Below is pseudo code demonstrating how the verification hash is calculated using a PayNow token. The Pay Now token can be obtained from the Fat Zebra Merchant Dashboard.

VerifyCard

Please make sure that the hash calculation is **only** the shared_secret, card token

Ruby

```ruby
shared_secret = "abc123" # Also known as your Pay Now token
card_token = "xyc12ce" # FatZebra issued token representing a debit or credit card

verification = hmac_md5(shared_secret, card_token)
# Expected value: 8cf7e7d50664d118c41a70b1ba22d916
```

Payments

Please make sure that reference (invoice #), amount (subunit) and currency are in the correct order as shown:

Ruby

```ruby
shared_secret = "abc123" # Also known as your Pay Now token

invoice_no = "INV4567" # Also known as reference
amount = "1000"
currency = "AUD"
hide_card_holder = true

hmac = HMAC::MD5.new(shared_secret)
data = [invoice_no, amount, currency]
data << [hide_card_holder] if hide_card_holder

# Data will be:
# INV4567:1000:AUD:true
verification = hmac_md5(shared_secret, data.join(":"))
# Expected value: c045c96c113ae660b91b60bd09feda20
# Hash will be 0a40877ca9f75152f27bf093af7fd44b if hide_card_holder is false
```

Updated 18 days ago

← Customer

PaymentMethod →

Did this page help you? Yes No

PaymentMethod

https://docs.fatzebra.com/docs/paymentmethod

PaymentMethod

The **PaymentMethod** object contains card details and is used in the following methods:

- renderPaymentsPage
- verifyCard

It supports both new cards (where the raw card number and details are provided) or existing cards (where a Fat Zebra card token is provided).

Pay with a new card

Attribute	Type	Description
`type`	string	Specify the following value: `card`
`data.number`	string	Card number
`data.holder`	string	Card holder name
`data.expiryMonth`	string	Expiry month 2 characters long Padded with a leading zero, e.g. `01` for January
`data.expiryYear`	string	Expiry year 4 characters long
`cvv`	string	CVV or CVC value on the card 3 or 4 digits depending on the card scheme

Example: card

```
{
  type: "card",
  data: {
    number: "4005550000000001",
    holder: "John Doe",
    expiryMonth: "03",
    expiryYear: "2022",
    cvv: "123"
  }
}
```

Pay with an existing card (card on file)

Attribute	Type	Description
`type`	string	Specify the following value: `card_on_file`
`data.token`	string	Card token. See here for more details on how to tokenize a card.

Example: card on file

```
{
  type: "card_on_file",
  data: {
    token: "fke8jmra"
  }
}
```

Updated about 4 years ago

PaymentIntent

HppSetupParams

Did this page help you? Yes No

HppSetupParams

https://docs.fatzebra.com/docs/hppsetupparams

HppSetupParams

The **HppSetupParams** object defines how the Hosted Payment Page (HPP) should be displayed via fatzebra.js, as well as setting the PaymentIntent.

`HppSetupParams` is used in the following methods:

- renderPaymentsPage

Attribute	Type	Description	Required/Optional	Default
`containerId`	string	The `id` of the HTML div element where the Hosted Payment Page iframe will be rendered	required	
`customer`	Customer	See Customer	required	
`paymentIntent`	PaymentIntent	See PaymentIntent	required	
`options.buttonText`	string	Specifies the text to be displayed for the submit button (e.g. Pay Now or Submit) Only applicable when hideButton is `false`	optional	empty
`options.cards`	string, comma separated	Specifies a whitelist or blacklist of accepted cards. If the string is prefixed with an exclamation mark `!` the cards listed will not be permitted Possible values are: `AMEX` , `JCB` , `VISA` , `MasterCard` , `Diners` , `Discover` , `China UnionPay` Examples ! `AMEX,JCB` - both AMEX and JCB are disabled `VISA,AMEX` - only enable VISA and AMEX Note if this field is present it is included in the end of the verification value string	optional	empty
`options.css`	string	HTTPS URL for external CSS. This must be a valid HTTPS URL that serves valid CSS. If specified,	optional	empty

Attribute	Type	Description	Required/Optional	Default
		`cssSignature` is also required.		
`options.cssSignature`	string	HMAC-MD5 of the css URL with the shared secret to sign the request	optional	empty
`options.hideButton`	boolean	Hides the button from the checkout form for when the checkout should be triggered by a purchase	optional	false
`options.hideCardHolder`	boolean	Hides the card holder entry field. If provided, needs to be in the verification hash calculation.	optional	false
`options.hideLogos`	boolean		optional	true
`options.logoUrl`	string	HTTPS URL for the merchant logo. This must be a valid HTTPS URL that serves an image file.	optional	empty
`options.showEmail`	boolean	Indicates whether to show and require the email field or not	optional	false
`options.showExtras`	boolean	Indicates whether to display the invoice number and the amount to the customer	optional	false
`options.tokenizeOnly`	boolean	When set to `true`, HPP will only tokenise the card and return the token via `fz.tokenization.success` event. 3DS/SCA and purchase will not run.	optional	false
`options.enableSca`	boolean	When set to `true`, 3DS/SCA will run after card tokenisation and before payment is made When `false`, a payment will be made after card tokenisation Please note that 3DS/SCA will not run when both `tokenizeOnly` and `enableSca` are set to `true`	optional	false
`options.challengeWindowSize`	string	The sizes are width x height in pixels of the window displayed in the cardholder browser. 2-character long string Available values are `'01'` : 250x400 `'02'` : 390x400 `'03'` : 500x600 `'04'` : 600x400 `'05'` : Full page (recommended)	optional	'05'

Example

```
const fz = new FatZebra({
  username: "MerchantXYZ"
```

```
});

fz.renderPaymentsPage({
  containerId: "fz-paynow",
  customer: {
    firstName: 'Captain',
    lastName: 'America',
    email: 'hello.world@fatzebra.com.au',
    address: '123 Australia Blvd.',
    city: 'Sydney',
    postcode: '2000',
    state: 'NSW',
    country: 'AU'
  },
  paymentIntent: {
    payment: {
      amount: 500,
      currency: "AUD",
      reference: "ref_123490"
    },
    verification: "ver_123480"
  },
  options: { // Hpp display options
    hideButton: false,
    hideLogos: true,
    enableSca: true
  }
});
```

Updated about 1 year ago

← PaymentMethod

VerifyCardParams →

Did this page help you? Yes No

VerifyCardParams

https://docs.fatzebra.com/docs/verifycardparams

VerifyCardParams

The **VerifyCardParams** object holds the values necessary to perform a 3DS2 check against a new or existing card.

Attribute	Type	Description	Required/Optional
`customer`	Customer	See Customer	required
`paymentIntent`	PaymentIntent	See PaymentIntent	required
`paymentMethod`	PaymentMethod	See PaymentMethod	required

JavaScript

```
const fz = new FatZebra({
  username: "MerchantXYZ"
});

fz.verifyCard({
  customer: {
    firstName: 'Captain',
    lastName: 'America',
    email: 'hello.world@fatzebra.com.au',
    address: '123 Australia Blvd.',
    city: 'Sydney',
    postcode: '2000',
    state: 'NSW',
    country: 'AU'
  },
  paymentIntent: {
    payment: {
      amount: 1000,
      currency: "AUD",
      reference: "ref_123490",
    },
    verification: "ver_123480"
  },
  paymentMethod: {
    type: "card",
    data: {
      number: "4111111111111111",
      holder: "John Doe",
      expiryMonth: "01",
      expiryYear: "2022",
      cvv: "123"
    }
  }
});
```

Updated over 3 years ago

← HppSetupParams

Events →

Did this page help you? Yes No

Events

https://docs.fatzebra.com/docs/events-1

Events

The **fatzebra.js** Javascript SDK will emit various events that your client-side Javascript code can subscribe to.

An event may be emitted along with a `data` payload, which is encapsulated into a CustomEvent object and can be retrieved via the detail property of the event.

List of fatzebra.js events

- Validation
- SCA
- Tokenization
- Payment

Successful Event Data Format

Attribute	Type	Description
`message`	string	message
`data`	object	data

example

```
{
  message: "xxx",
  data: {
   ...
  }
}
```

Error Event Data Format

Attribute	Type	Description
`errors`	array of string	error messages
`data`	object	This could be `null` if no data is available.

example

```
{
  errors: [
    "xxx",
    ...
  ],
  data: {
   //
  }
}
```

Subscribing to an event

Example

```
const fz = new FatZebra({
  username: "MerchantXYZ"
});
```

```
// Handle errors related to general errors. For example, client-side validations.
fz.on('fz.validation.error', function(event) {
  // ...
});

// Handle errors related to payment processing. HPP only.
fz.on('fz.payment.error', function(event) {
  // prompt error to customer
  console.log('payment errors: ' + JSON.stringify(event.detail.errors));
});

// Handle successful payment processing. HPP only.
fz.on('fz.payment.success', function(event) {
  console.log('payment data: ' + JSON.stringify(event.detail.data));
});
```

Updated about 4 years ago

← VerifyCardParams

Validation →

Did this page help you? Yes No

Validation

https://docs.fatzebra.com/docs/validation

Validation

> Currently validation is only imposed on methods exposed by fatzebra.js. See Methods.
>
> Support for propagating Hosted Payments Page (HPP) form validation errors is on the road map.

Event	Description	Applicable Methods
`fz.validation.error`	Client-side validation errors occurred while calling fatzebra.js methods.	verifyCard, renderPaymentsPage, checkout

Examples of validation errors include:

Required fields

fz.validation.error data payload - required field

```
{
  errors: [
    "/paymentIntent/payment/amount is required",
    ...
  ],
  data: null
}
```

Data type invalid

fz.validation.error data payload - data type invalid

```
{
  errors: [
    "/paymentIntent/payment/amount should be an integer",
    ...
  ],
  data: null
}
```

Updated about 3 years ago

Events

SCA

Did this page help you? Yes No

SCA

https://docs.fatzebra.com/docs/sca

SCA

Event	Description	Applicable Methods
`fz.sca.success`	Emitted when 3DS validation completes successfully. The liability shifts from merchant to the card issuer. Please note: The edge case is when the enrolled field (ver) comes back as 'N' (3DS success - Not Enrolled), which means "No, Bank is not participating in 3-D Secure protocol". If the Enrolled value is equal to N, then the Consumer is NOT eligible for Authentication (No liability shift). Most merchants choose to continue with these transactions, because otherwise in many cases it would be stopping customers from making legitimate transactions.	renderPaymentsPage verifyCard
`fz.sca.error`	Emitted when 3DS enrolment or validation failed. The payment attempt is deemed risky and merchant shall not proceed with the payment.	renderPaymentsPage verifyCard

fz.sca.success Data Payload

Attribute	Type	Description
`aav`	string	Account Authentication Value. Unique 32-character transaction token for a 3D Secure transaction. For Mastercard Identity Check, the AAV is named the UCAF. For Visa Secure, the AAV is named the CAVV.
`cavv`	string	Cardholder Authentication Verification Value. A Base64-encoded string sent back with Visa Secure-enrolled cards that specifically identifies the transaction with the issuing bank and Visa. Standard for collecting and sending AAV data for Visa Secure transactions.
`par`	string	Payer Authentication Response. Compressed, Base64-encoded response from the card-issuing bank
`sli`	string	The Security Level Indicator for 3DS transactions
`xid`	string	String used by both Visa and Mastercard which identifies a specific transaction on the Directory Servers. This string value should remain consistent throughout a transaction's history.
`ucaf`	string	Universal Cardholder Authentication Field. Mastercard only.
`ver`	string	3DS enrolment status.
`directoryServerTxnId`	string	Directory server transaction Id
`threedsVersion`	string	3DS version used for verifying the intended payment.

fz.sca.success data payload

```
{
  message: "xxx",
  data: {
    aav: "xxx",
    cavv: "xxx",
```

```
    par: "xxx",
    sli: "xxx",
    xid: "xxx",
    ucaf: "xxx",
    ver: "xxx",
    directoryServerTxnId: "xxx",
    threedsVersion: "xxx",
  }
}
```

fz.sca.error Data Payload

Attribute	Type	Description
`errorCode`	string	Error code for specific 3DS/SCA failure scenario. errorCode will be not be present in the event of request timeout or server error See below for error code mapping.

fz.sca.error data payload

```
// Request timeout, server error, etc. Prompt user to retry.
{
  errors: ["xxx"],
  data: null
}

// 3DS2 authentication failed. The card is deemed risky by issuer.
{
  errors: ["xxx"],
  data: {
    errorCode: "xxx"
  }
}
```

3DS/SCA Error Code Mapping

Error Code	Description
`001`	**Bypassed authentication** This is related to a feature that is **currently not available**. We will look at providing merchants the ability to configure rules to determine whether authentication is required for a transaction. -- No liability shift
`002`	**Authentication not available on lookup** Unable to verify 3DS enrolment status with the issuer. -- No liability shift
`003`	**Unavailable frictionless authentication** 3DS authentication is unavailable with the issuer. -- No liability shift
`004`	**Unsuccessful frictionless authentication** The issuer deems the transaction as risky -- No liability shift
`005`	**Rejected frictionless authentication** -- No liability shift
`006`	**Unsuccessful step-up authentication** Unsuccessful authentication due to failed OTP (one time password) challenge. -- No liability shift

Error Code	Description
`007`	**Unavailable step-up authentication** The card holder is enrolled for 3DS, but authentication is not available with that issuer. -- No liability shift

Updated over 1 year ago

← Validation

Tokenization →

Did this page help you? Yes No

Tokenization

https://docs.fatzebra.com/docs/tokenization

Tokenization

All cards entered into the Hosted Payment Page are tokenized first before 3DS verification and payment transaction.

Event	Description	Applicable Methods
`fz.tokenization.success`	Emitted when card is successfully tokenized.	renderPaymentsPage
`fz.tokenization.error`	Emitted when card tokenization fails.	renderPaymentsPage

fz.tokenization.success Data Payload

Attribute	Type	Description
`token`	string	Card token

fz.tokenization.success data payload

```
{
  message: "xxx",
  data: {
    token: "xxx",
  }
}
```

fz.tokenization.error Data Payload

Attribute	Type	Description
N/A	N/A	N/A

fz.tokenization.error data payload

```
{
  errors: ["xxx"],
  data: null
}
```

Updated about 4 years ago

SCA

Payment

Did this page help you? Yes No

Payment

https://docs.fatzebra.com/docs/payment

Payment

Event	Description	Applicable Methods
`fz.payment.success`	Emitted when payment is processed successfully.	renderPaymentsPage
`fz.payment.error`	Emitted when a payment transaction failed.	renderPaymentsPage

fz.payment.success data payload

fz.payment.success data payload


```
{
  "message": "Payment successful.",
  "data": {
    "transactionId": "40057-P-F7R7M9Q6",
    "responseCode": "00",
    "message": "Approved",
    "amount": 100,
    "currency": "AUD",
    "reference": "sgc99pycds20i97q",
    "cardNumber": "400000XXXXXX1091",
    "cardHolder": "XXX",
    "cardExpiry": "2023-12-31",
    "cardType": "VISA",
    "verification": "0aa8154b8cf0fe4168af0e3d0d752d10"
  }
}
```


It is recommended that you verify the returned data against the calculated verification (see below) before consumption.

fz.payment.error data payload

fz.payment.error data payload


```
{
  "message": "Refer to Card Issuer",
  "data": {
    "transactionId": "40057-P-F7R7M9Q6",
    "responseCode": "05",
    "message": "Declined",
    "amount": 100,
    "currency": "AUD",
    "reference": "sgc99pycds20i97q",
    "cardNumber": "400000XXXXXX1091",
    "cardHolder": "XXX",
    "cardExpiry": "2023-12-31",
    "cardType": "VISA",
    "verification": "0aa8154b8cf0fe4168af0e3d0d752d10"
  }
}
```


Transaction Data

Attribute	Type	Description
`transactionId`	string	Payment transaction id
`responseCode`	string	
`message`	string	

Attribute	Type	Description
`amount`	number	amount in subunit. e.g. AUB $10.25 -> 1025
`currency`	string	currency code. e.g. AUD
`reference`	string	payment reference or invoice #.
`cardNumber`	string	masked card number
`cardHolder`	string	card holder name
`cardExpiry`	string	card expiry date. yyyy-mm-dd
`cardType`	string	card type. e.g. VISA, MasterCard, etc
`verification`	string	calculated verification hash.

Verification hash calculation

values = [transactionId]:[responseCode]:[message]:[amount]:[currency]:[reference]:[cardNumber]:[cardHolder]:[cardExpiry]:[cardType]

shared_secret =

verification = MD5(values, shared_secret)

For example:

values = "40057-P-F7R7M9Q6","00":"Declined": 100 :"AUD":"sgc99pycds20i97q":"400000XXXXXX1091":"XXX":"2023-12-31":"VISA":"0aa8154b8cf0fe4168af0e3d0d752d10"

shared_secret = "123"

verification = MD5(values, shared_secret)

Updated over 3 years ago

← Tokenization

Form Validation →

Did this page help you? Yes No

Form Validation

https://docs.fatzebra.com/docs/form-validation

Form Validation

> Hosted Payments Page form validation errors are propagated to the parent frame (merchant website) via fz.form_validation.error event.

Event	Description	Applicable Methods
`fz.form_validation.error`	Hosted Payments Form validation errors	renderPaymentsPage

fz.form_validation.error Data Payload

Example

JSON

```
{
  "message": "Form was not valid",
  "errors": [
    "Expiry Date is required",
    "Security Code (CVV) is required"
  ],
  "data": {
    "payment_request_expiry_date": [
      {
        "errorCode": "hpp_10401",
        "message": "Expiry Date is required"
      }
    ],
    "payment_request_security_code": [
      {
        "errorCode": "hpp_10301",
        "message": "Security Code (CVV) is required"
      }
    ]
  }
}
```

- *errors* contains an array of messages associated with the form validation errors.
- *data* provides details around the affected fields and the associated errors, along with error codes. Developers can take advantage of the error code in order to customise the user experience.

Validation Rules

Field	Rule	Error Code
payment_request_card_holder	required	hpp_10101
payment_request_card_holder	max length * 50 characters	hpp_10102
payment_request_card_holder	pattern * alphanumeric only	hpp_10103
payment_request_card_number	required	hpp_10201
payment_request_card_number	min length * 13 characters	hpp_10202
payment_request_card_number	max length * 19 characters	hpp_10203

Field	Rule	Error Code
payment_request_card_number	pattern * valid credit card number pattern	hpp_10204
payment_request_card_number	supported card types * **applicable only when `options.cards` is provided.** See HppSetupParams for details	hpp_10205
payment_request_security_code	required	hpp_10301
payment_request_security_code	min length * 3 characters	hpp_10302
payment_request_security_code	max length *4 characters for AMEX & JCB* 3 characters for everything else.	hpp_10303
payment_request_security_code	pattern * numeric only	hpp_10304
payment_request_expiry_date	required	hpp_10401
payment_request_expiry_date	valid expiry date * greater than today	hpp_10402
payment_request_email	required * **applicable only when `options.showEmail` is provided.** See HppSetupParams for details	hpp_10501
payment_request_email	pattern * valid email format	hpp_10502
payment_request_email	max length * 100 characters	hpp_10503

Updated about 3 years ago

← Payment

Overview →

Did this page help you? Yes No

Overview

https://docs.fatzebra.com/docs/3ds2-overview

Overview

3DS2 allows you to authenticate your transactions to reduce fraud and shift liability

3DS2 is a payments industry standard that promotes secure, frictionless consumer authentication for online payments. 3DS2 allows you to reduce fraud by providing methods to verify the authenticity of online payments.

The steps required to implement 3DS2 using Fat Zebra will depend on:

1. If you have already have an MPI (Merchant Plug-in) or require Fat Zebra to act as your MPI
2. Your PCI level
3. Your preferred integration method

MPI

I already have an MPI

If you have an existing MPI integration you will need to follow your MPI's instructions in order to authenticate a payment transaction. After authentication your MPI will provide you with 3DS data which may include:

- Cryptogram, CAV, CAVV or AAV
- Ecommerce Indicator (ECI) or Security Level Indicator (SLI)
- PARes
- VERes

You will then need to set the 3D Secure fields on your Fat Zebra Purchase or Refund API calls in order to complete a 3DS2 transaction.

I don't have an MPI

Fat Zebra can act as your MPI. Please contact your Fat Zebra representative for more information on how to add Fat Zebra's 3DS2 MPI to your account.

Using Fat Zebra's MPI, there are various integration methods to choose from depending on your PCI level:

Integration Method	Description	3DS Integration Guide
Hosted Payments Page	**PCI SAQ A required** Your website uses Fat Zebra's payment form through the Fat Zebra Hosted Payments Page iframe. Payment processing is entirely outsourced to FatZebra, meaning you do not need to handle any raw card data.	Using Hosted Payments Page
Direct Post	**PCI A-EP required** Your website hosts the payment form and you are responsible for collecting card details on front end code you host. Card data exchange with Fat Zebra will take place via an AJAX call from your frontend directly to Fat Zebra.	Using Custom Payments Form
API	**PCI SAQ D required** Your website hosts the payment form and you are responsible for collecting card details in both your backend and frontend.	Using Custom Payments Form

Integration Method	Description	3DS Integration Guide
	Card data exchange with Fat Zebra will take place via API calls from your backend system to Fat Zebra.	

Updated about 4 years ago

← Form Validation

Using Hosted Payments Page →

Did this page help you? Yes No

Using Hosted Payments Page

https://docs.fatzebra.com/docs/using-hosted-payments-page

Using Hosted Payments Page

> 3DS2 via the Hosted Payments is only supported via **fatzebra.js**. If you are currently using a direct integration with the Hosted Payments Page by creating your own iframe and Hosted Payments Page URL you will need to switch over to **fatzebra.js** in order to implement 3DS2

Overview

Fat Zebra's Hosted Payments Page (HPP) provides a seamless way to perform 3DS2 checks on either new or existing cards before a purchase is made.

When your customers submit a payment, the Fat Zebra Hosted Payments Page form will perform the following operations behind the scenes and emit javascript events so that you can react appropriately:

1. Tokenise the card details entered into the Hosted Payments Page
2. Perform a 3DS2 check against the card (Note: the `enableSca` URL parameter must be set to **true** in order for this to happen, more details in Step 5 below)
3. Upon a successful 3DS2 check, HPP will then make a purchase call with the 3DS2 results to process the payment. If the 3DS2 check is unsuccessful, fatzebra.js will emit an event that you can react to.

> 📘 Before completing the steps below, speak to the Fat Zebra support team to have 3DS2 enabled on your account

Step 1 - Obtain an OAuth access token

Follow the steps in Obtain an OAuth token to obtain an access token for your checkout session.

Step 2 - Include fatzebra.js on your page

HTML

```
<script type="text/javascript" src="https://cdn.pmnts-sandbox.io/sdk/v1/fatzebra.js"></script>
```

Step 3 - Initialise the FatZebra JS SDK

Initialize the `FatZebra` object in the footer of your page:

JavaScript

```
// Initialize the FatZebra object in the footer of your page
<script>
const fz = new FatZebra({
  username: "<YOUR MERCHANT USERNAME>"
});

...
</script>
```

Note: your merchant username can be obtained from the Fat Zebra Merchant Dashboard.

Step 4 - Subscribe to fatzebra.js events

fatzebra.js will emit various javascript events. Subscribe to the ones relevant to you.

Available events include:

- Validation - for validation errors
- Tokenization - for success or failure of the tokenization of the card data
- SCA - for success or failure of the 3DS2 checks
- Payment - for success or failure of the payment

JavaScript

```
// Handle validation related errors, e.g. client-side validation
fz.on('fz.validation.error', function(event) {
  // ...
});

// Receive a FZ card token
fz.on('fz.tokenization.success', function(event) {
  // If required you can save the card token in your backend
});

// Handle errors related to payment processing
fz.on('fz.payment.error', function(event) {
  // Show an error message to the customer
});

fz.on('fz.payment.success', function(event) {
  // checkout ends
});

// Handle errors related to SCA
fz.on('fz.sca.error', function(event) {
  // Show an error message to the customer
});
```

Step 5 - Load Hosted Payments Page

enableSca: **true**

See HppSetupParams for HPP options.

Note: The customer object is only required when enableSca is set to **true**.

JavaScript

```
fz.renderPaymentsPage({
  containerId: "fz-paynow",
  customer: {
    firstName: 'Captain',
    lastName: 'America',
    email: 'hello.world@fatzebra.com.au',
    address: '123 Australia Blvd.',
    city: 'Sydney',
    postcode: '2000',
    state: 'NSW',
    country: 'AU'
  },
  paymentIntent: {
    payment: {
      amount: 500,
      currency: "AUD",
      reference: "ref_123490"
    },
    verification: 'ver_123480' // made up value
  },
  options: { // Hpp display options
    hideButton: false,
    hideLogos: true,
    enableSca: true
  }
});

// Register click event handler to trigger Hosted Payment Page payment flow if
// you have a custom checkout button.
const handler = function() {
  fz.checkout();
}

document.getElementById('checkoutButton').addEventListener('click', handler);
```

Your payment page is now ready for 3DS2 checks. When your customers enter card details and submit a payment, 3DS2 checks will now occur and either prompt the customer for additional verification details (challenge flow) or submit the payment without a customer prompt (frictionless flow).

Updated over 3 years ago

WHAT'S NEXT

Testing →

Go-Live →

Did this page help you? Yes No

Using Custom Payments Form

https://docs.fatzebra.com/docs/using-custom-payments-form

Using Custom Payments Form

Overview

If you collect card details using your own payments form, you can perform 3DS2 checks using fatzebra.js.

> 📘 Before completing the steps below, speak to the Fat Zebra support team to have 3DS2 enabled on your account

Step 1 - Obtain an OAuth access token

Follow the steps in Obtain an OAuth token to obtain an access token for your checkout session.

Step 2 - Include fatzebra.js on your page

HTML

```
<script type="text/javascript" src="https://cdn.pmnts-sandbox.io/sdk/v1/fatzebra.js"></script>
```

Step 3 - Initialise the FatZebra SDK

JavaScript

```
// Initialize the FatZebra object in the footer of your page
<script>
const fz = new FatZebra({
  username: "<YOUR MERCHANT USERNAME>"
});

...
</script>
```

Note: your merchant username can be obtained from the Fat Zebra Merchant Dashboard.

Step 4 - Subscribe to fatzebra.js events

fatzebra.js will emit various javascript events. Subscribe to the ones relevant to you.

Available events:

- Tokenization - for success or failure of the tokenization of the card data
- SCA - for success or failure of the 3DS2 checks

JavaScript

```
// Handle validation related errors, e.g. client-side validation
fz.on('fz.validation.error', function(event) {
  // ...
});

// Receive the result of the 3DS2 check.
fz.on('fz.sca.success', function(event) {
  // Obtain 3DS2 results which will be used to make a purchase in the backend
})

// Handle errors related to SCA
fz.on('fz.sca.error', function(event) {
  // Show an error message to the customer
});
```

Step 5 - Perform 3DS2 checks on the card

Call the fatzebra.js `verifyCard` method to perform 3DS2 on a card.

See verifyCard and VerifyCardParams for more details.

JavaScript

```
// Call this function when the checkout button on your page is clicked
fz.verifyCard({
  customer: {
    firstName: 'Captain',
    lastName: 'America',
    email: 'hello.world@fatzebra.com.au',
    address: '123 Australia Blvd.',
    city: 'Sydney',
    postcode: '2000',
    state: 'NSW',
    country: 'AU'
  },
  paymentIntent: {
    payment: {
      amount: 1000,
      currency: "AUD",
      reference: "ref_123490",
    },
    verification: "ver_123480" // made up value
  },
  paymentMethod: {
    type: "card",
    data: {
      number: "4111111111111111",
      holder: "John Doe",
      expiryMonth: "01",
      expiryYear: "2022",
      cvv: "123"
    }
  }
});
```

Step 6 - Frictionless or challenge flow presented to customer

The fatzebra.js `verifyCard` will send the payment and customer details onto the issuing bank of the cardholder. The issuing bank will then determine if:

a) The customer should be presented with a 3DS2 challenge to prove they are the cardholder. This normally involves the customer receiving a one-time pin (OTP) on their phone and then be required to enter the OTP into a modal dialog on the payment page. The fatzebra.js `verifyCard` method will automatically display this modal dialog, no additional code is required on your end.

b) If the issuing bank determines that the customer is low risk they may allow the customer to go through a "frictionless" flow, whereby the customer does not need to complete any additional steps (such as an OTP).

Step 7 - Obtain 3DS check result

After the customer has completed the frictionless or challenge flow, fatzebra.js will emit one of the following events:

a) `fz.sca.success` : when 3DS validation completes successfully
b) `fz.sca.error` : when 3DS enrolment or validation failed

In the event of a successful 3DS check, the `fz.sca.success` event will contain 3DS result data (`cavv` , `sli` , etc.) that can be sent in your backend purchase or refund call. The presence of valid 3DS result data in your Purchase or Refund API will enable a liability shift to the issuer for fraudulent transactions.

> 🚧 A separate check with successful 3DS result data is required for each payment you wish to make. The same 3DS result data cannot be used in multiple purchases.

Step 8 - Make a purchase using 3DS result data

Provide the 3DS result data from the `fz.sca.success` event in the request payload `extra` field when making a Purchase or Refund API call. Please note that these API calls require `snake_case` keys, i.e. `directory_server_txn_id` , not `directoryServerTxnId`

See 3DS extra payload for more information.

JSON

```
{
    "amount": 10,
    "currency": "AUD",
    "card_token": "syupxv5b1tqv4bqdsazr",
    "reference": "GfNStL8XJ1111",
    "customer_ip": "111.222.111.123",
    "extra": {
      "sli": "05",
      "cavv": "kAMRDwUADQVXiAPovxs+SQNhfGpb",
      "xid": "",
      "ver": "Y",
      "par": "Y",
      "directory_server_txn_id": "a8a35bcf-21d5-4f55-995a-a4b5e60d356d",
      "threeds_version": "2"
  }
}
```


Updated over 3 years ago

WHAT'S NEXT

Testing →

Go-Live →

Did this page help you? Yes No

Testing

https://docs.fatzebra.com/docs/testing

Testing

A 3DS check may end up with one of the following outcomes.

- 3DS check successful
  `fz.sca.success` event is triggered by fatzebra.js. If you own your own payments form and use the `verifyCard` method of fatzebra.js to perform 3DS checks, 3DS result data will be provided in the `fz.sca.success` event payload. Using this 3DS result data in your Purchase or Refund API call will qualify that transaction for liability shift.
- Please note: The edge case is when the enrolled field (ver) comes back as 'N' (3DS success - Not Enrolled), which means "No, Bank is not participating in 3-D Secure protocol". If the Enrolled value is equal to N, then the Consumer is NOT eligible for Authentication (No liability shift).
- 3DS check unsuccessful
  `fz.sca.error` event is triggered by fatzebra.js. An error code will be returned in the event payload indicating the reason for the error. This could be due to the card issuer not support 3DS2 or if the cardholder was unable to verify their card when prompted with a 3DS challenge flow (e.g. they entered the wrong one-time pin that was sent to their mobile device).
- Server Error or timeout.
  `fz.sca.error` event is triggered by fatzebra.js. No error code will be returned in the event payload. In this event we recommend that you prompt your users to retry the 3DS check/payment.

Test cards

FatZebra provides a number of test cards to simulate various 3DS scenarios in the Sandbox environment. Each of the following card numbers will simulate a predetermined outcome:

Successful 3DS2 check

Scenario	Cards
Successful frictionless authentication Cardholder being authenticated by their Card Issuer -- Liability shift	Visa: 4000000000001000 Mastercard: 5200000000001005 Amex: 340000000001007
Attempts processing frictionless authentication Cardholder is enrolled in 3DS however the Issuer is not supporting the program, resulting in a stand-in authentication experience -- Liability shift	Visa: 4000000000001026 Mastercard: 5200000000001021 Amex: 340000000001023
Successful step-up authentication Successful OTP challenge -- Liability shift	Visa: 4000000000001091 Mastercard: 5200000000001096 Amex: 340000000001098

fz.sca.success sample payload

JSON


```
{
  "message": "FatZebra.3DS: 3DS success - Successful Frictionless Authentication",
  "data": {
    "cavv": "xxx",
    "par": "xxx",
    "sli": "xxx",
    "xid": "xxx",
    "ver": "xxx",
    "directoryServerTxnId": "xxx",
```

```
    "threedsVersion": "xxx"
  }
}
```

Unsuccessful 3DS check

Scenario	Card	Error code
Unsuccessful enrolment check Enrolment failed by card issuer -- No liability shift	Visa: 4000000000002446 Mastercard: 5200000000002037 Amex: 340000000002732	002
Unsuccessful frictionless authentication Authentication failed by card issuer without challenge -- No liability shift	Visa: 4000000000001018 Mastercard: 5200000000001013 Amex: 340000000001015	004
Unavailable frictionless authentication Authentication is unavailable from the issuer at the current time. -- No liability shift	Visa: 4000000000001034 Mastercard: 5200000000001039 Amex: 340000000001031	003
Rejected frictionless authentication Authentication rejected by the issuer. -- No liability shift	Visa: 4000000000001042 Mastercard: 5200000000001047 Amex: 340000000001049	005
Authentication not available on lookup Unable to verify 3DS enrolment status with the issuer due to system error. -- No liability shift	Visa: 4000000000001059 Mastercard: 5200000000001054 Amex: 340000000001056	002
Server Error -- No liability shift	Visa: 4000000000001067 Mastercard: 5200000000001062 Amex: 340000000001064	-
Timeout -- No liability shift	Visa: 4000000000001075 Mastercard: 5200000000001070 Amex: 340000000001072	-
Bypassed authentication This is related to a feature that is **currently not available**. We will look at providing merchants the ability to configure rules to determine whether authentication is required for a transaction. -- No liability shift	Visa: 4000000000001083 Mastercard: 5200000000001088 Amex: 340000000001080	001
Unsuccessful step-up authentication Unsuccessful authentication due to failed OTP (one time password) challenge. -- No liability shift	Visa: 4000000000001109 Mastercard: 5200000000001104 Amex: 340000000001106	006
Unavailable step-up authentication The card holder is enrolled for 3DS, but authentication is not available with that issuer.	Visa: 4000000000001117 Mastercard:	007

Scenario	Card	Error code
-- No liability shift	5200000000001112 Amex: 340000000001114	

fz.sca.error sample payload

fz.sca.error

```
{
  "errors": [
    "FatZebra.3DS: 3DS error - Rejected Frictionless Authentication"
  ],
  "data": {
    "errorCode": "005"
  }
}
```

Updated about 1 year ago

← Using Custom Payments Form

Branding Requirements →

Did this page help you? Yes No

Branding Requirements

https://docs.fatzebra.com/docs/branding-requirements

Branding Requirements

Each card scheme recommends merchants to display their 3DS2 product logos on their payment pages where possible. These product logos include:

- Verified by Visa
- Mastercard ID Check
- Amex SafeKey

If you are integrating 3DS2 using Fat Zebra's Hosted Payments Page, please note that the Hosted Payments Page does not display these logos for you in order to give you control of the placement of these logos.

Updated about 4 years ago

← Testing

Go-Live →

Did this page help you? Yes No

Go-Live

https://docs.fatzebra.com/docs/go-live

Go-Live

Before going live in production, please contact the Fat Zebra support team to enable production 3DS2 access on your account.

Updated about 4 years ago

Branding Requirements

3DS2 Outcomes

Did this page help you? Yes No

3DS2 Outcomes

https://docs.fatzebra.com/docs/3ds2-outcomes

3DS2 Outcomes

3DS2 events can result in a variety of outcomes

Outcome	Description
Frictionless	Customer is not made aware of the successful 3DS2 event.
Challenged	The customer is challenged to prove legitimate access to the payment method. A pop up window is displayed to the customer, asking them to authenticate their credentials. This is usually completed via SMS.
Unsuccessful	Rejected by the issuer
Unavailable	3DS2 service is unavailable or the merchant has not been registered correctly for 3DS2.
Abandoned	The customer has abandoned their 3DS authentication by failing to enter in the passcode and navigating away or letting the session timeout.
Rejected	Authentication rejected by the issuer.
Invalid Scenario	This can occur when an older version of 3DS is used by the card issuer.

Updated 12 months ago

Go-Live

Overview

Did this page help you? Yes No

Overview

https://docs.fatzebra.com/docs/merchant-dashboard-user-guide

Overview

The Merchant Dashboard allows you to view information about transactions that have occurred through your merchant account. Through this dashboard you can:

- Create new purchases
- Refund or retry existing purchases
- Authorize a payment for a future date
- View reports for these transactions
- Securely store credit card information for customers
- Setup payment plans for your customers

Look for more documentation in the sidebar of this site, under "Merchant Dashboard User Guide".

Updated about 2 years ago

← 3DS2 Outcomes

Transactions →

Did this page help you? Yes No

Transactions

https://docs.fatzebra.com/docs/transactions

Transactions

Transactions in the Merchant Dashboard include Purchases, Authorizations and Refunds.

Updated about 2 years ago

Overview

Individual Transaction Search

Did this page help you? Yes No

Individual Transaction Search

https://docs.fatzebra.com/docs/user-guide

Individual Transaction Search

1. Select Transaction > Purchases
2. Select a Date Range to filter purchases by a given date range.
3. Enter search term in the Search text box. The search options can include ID, Reference, Card Holder and Card Number
4. Use the "Result" and "Currency" filters to only return transactions that match those particular filters.

Updated about 2 years ago

← Transactions

Refund A Transaction →

Did this page help you? Yes No

Refund A Transaction

https://docs.fatzebra.com/docs/refund-a-transaction

Refund A Transaction

1. Select Transaction > Purchases
2. Select Refund from the transaction drop down menu.

5. Select the tab to 'Refund full amount' or 'Refund partial amount'
6. Enter a Description (Optional) and a Unique Reference
7. Select the Refund button to complete the transaction

Updated about 2 years ago

Individual Transaction Search

Repeat A Transaction

Did this page help you? Yes No

Repeat A Transaction

https://docs.fatzebra.com/docs/repeat-a-transaction

Repeat A Transaction

1. Select Transaction > Purchases
2. Select Repeat from the transaction drop down menu.

5. Complete the payment form by entering the Amount, Description (Optional), a Unique Reference, and the Card details.
6. Select the Charge Card button to complete the transaction

Updated about 2 years ago

Refund A Transaction

Email or Download Receipt

Did this page help you? Yes No

Email or Download Receipt

https://docs.fatzebra.com/docs/email-or-download-receipt

Email or Download Receipt

1. Select Transaction > Purchases
2. Select ‘Download Receipt’ or ‘Email Receipt’ from the transaction drop down menu.

5. If emailing a receipt, enter the email address and select the ‘Send’ button.

Updated about 2 years ago

Repeat A Transaction

Virtual Terminal

Did this page help you? Yes No

Virtual Terminal

https://docs.fatzebra.com/docs/virtual-terminal

Virtual Terminal

1. Select Virtual Terminal
2. Complete the payment form by entering the Amount, Description (Optional), a Unique Reference, and the Card details.
3. Select the Charge Card button to complete the transaction

Please find an example of our Virtual Terminal below.

Our Virtual Terminal provides an option to 'Authorization Only'. This transaction type places a hold on the funds, but does not debit the card. You will need to capture the funds within 4-7 days if you proceed with this option.

Updated over 2 years ago

Email or Download Receipt

Disputes

Did this page help you? Yes No

Disputes

https://docs.fatzebra.com/docs/disputes-1

Disputes

For Merchants acquiring through our TPP, Disputes (Credit card chargebacks) can be viewed, contested and managed via the Dashboard, under Transactions > Disputes.

Dispute Statues are as below:

Open - The dispute is open and has yet to be contested

Contested - The dispute was contested by the merchant

Accepted - The dispute was accepted by the merchant

Won - The dispute was successfully contested

Lost - The dispute was unsuccessfully contested

Expired - The response due date passed without the dispute being contested

Individual Dispute Search

1. Select Transaction > Disputes
2. Select a Date Range to filter disputes by a given date range.
3. Enter transaction id in the Search text box.
4. Select status, order by and sort by options.

Managing Disputes

Disputes can be contested or accepted, by clicking on the Transaction ID of the dispute in question, then clicking on either 'Upload Evidence' to contest the dispute, or 'Accept dispute' to accept the dispute.

Please note, uploaded evidence needs to be in PDF format.

Dashboard users can also subscribe to Chargeback Notification emails, via the Reports > Subscribers section of the Merchant Dashboard, this will give the user an email notification each time a dispute is raised.

Receive reports by email

https://docs.fatzebra.com/docs/email-reports

Receive reports by email

1. Select Reports > Subscribers
2. Select New Subscriber from the Actions menu
3. Enter the Recipient name and email address
4. Select each report that you would like the subscriber to receive. Please find the options below
5. Emails are sent out every morning at 8am to their subscribers.

5. Select the Add Subscriber button

Updated about 2 years ago

← Disputes

Transactions Report →

Did this page help you? Yes No

Transactions Report

https://docs.fatzebra.com/docs/download-transactions

Transactions Report

1. Select Reports > Transactions
2. Select a Date Range
3. Select Download Detailed Report from the Actions menu

Please note that there is a 'Based on settlement date' checkbox option. If this option is not selected, you will receive transactions from and up to midnight within the selected date range.

Updated about 2 years ago

← Receive reports by email

Settlement Summary Report →

Did this page help you? Yes No

Settlement Summary Report

https://docs.fatzebra.com/docs/settlement-summary-report-1

Settlement Summary Report

1. Select Reports > Settlements Summary
2. Select a Date Range

You will be able to view a list of the daily settlement amounts within your date range. Amex and Diners will appear in separate columns to allow you to reconcile these amounts when they are settled into your bank account.

The settlement summary report provides a 'View' option that provides a breakdown of transactions by card type. You can also download an individual transaction report when viewing a settlement day.

Updated over 2 years ago

← Transactions Report

All Merchant Report →

Did this page help you? Yes No