Coinbase Pro

- REST API
- FIX API
- Websocket Feed

# Introduction

Welcome to Coinbase Pro trader and developer documentation. These documents outline exchange functionality, market details, and APIs.

APIs are separated into two categories: trading and feed. Trading APIs require authentication and provide access to placing orders and other account information. Feed APIs provide market data and are public.

> By accessing the Coinbase Pro Market Data API, you agree to be bound by the Market Data Terms of Use.

# Upcoming Changes

8/23/18 - Query parameters on `/fills` will now be required. more info

> Changes are deployed at or near `1pm *PT*`.

# General

Market overview and general information.

# Matching Engine

Coinbase Pro operates a continuous first-come, first-serve order book. Orders are executed in price-time priority as received by the matching engine.

## Self-Trade Prevention

Self-trading is not allowed on Coinbase Pro. Two orders from the same user will not fill one another. When placing an order, you can specify the self-trade prevention behavior.

**DECREMENT AND CANCEL**

The default behavior is decrement and cancel. When two orders from the same user cross, the smaller order will be canceled and the larger order size will be decremented by the smaller order size. If the two orders are the same size, both will be canceled.

#### CANCEL OLDEST

Cancel the older (resting) order in full. The new order continues to execute.

#### CANCEL NEWEST

Cancel the newer (taking) order in full. The old resting order remains on the order book.

#### CANCEL BOTH

Immediately cancel both orders.

#### NOTES FOR MARKET ORDERS

When a `market` order using `dc` self-trade prevention encounters an open limit order, the behavior depends on which fields for the market order message were specified. If `funds` and `size` are specified for a buy order, then `size` for the market order will be decremented internally within the matching engine and `funds` will remain unchanged. The intent is to offset your target size without limiting your buying power. If `size` is not specified, then funds will be decremented. For a market sell, the size will be decremented when encountering existing limit orders.

---

## Price Improvement

Orders are matched against existing order book orders at the price of the order on the book, not at the price of the taker order.

#### EXAMPLE

User A places a Buy order for 1 BTC at 100 USD. User B then wishes to sell 1 BTC at 80 USD. Because User A's order was first to the trading engine, they will have price priority and the trade will occur at 100 USD.

---

## Order Lifecycle

Valid orders sent to the matching engine are confirmed immediately and are in the `received` state. If an order executes against another order immediately, the order is considered done. An order can execute in part or whole. Any part of the order not filled immediately, will be considered `open`. Orders will stay in the `open` state until canceled or subsequently filled by new orders. Orders that are no longer eligible for matching (filled or canceled) are in the `done` state.

---

# Fees

---

## Trading Fees

Coinbase Pro operates a maker-taker model. Orders which provide liquidity are charged different fees from orders taking liquidity. The fee is assessed as a percentage of the match amount (price * size). Maker and taker fees are calculated hourly based on the user's 30d USD-equivalent trading volume:

| User 30 day USD volume | Taker fee | Maker fee |
|---|---|---|
| $0 - $10m | 0.30 % | 0 % |
| $10m - $100m | 0.20 % | 0 % |

| User 30 day USD volume | Taker fee | Maker fee |
|---|---|---|
| $100m+ | 0.10 % | 0 % |

#### HOW TAKER FEES ARE CALCULATED

Your taker fee is based upon total USD trading volume across all Order Books over the trailing 30 day period. For example, a purchase of 1 BTC for $10,000 on the BTC-USD book will count as $10,000 towards your 30 day USD volume.

#### HOW VOLUMES ARE CALCULATED ON NON-USD BOOKS

Transactions made on non-USD books are converted to USD based on the most recent fill price on the corresponding USD book. For example, a purchase of 1 ETH for 0.1 BTC on the ETH-BTC book will count as the most recent fill price of 1 ETH on the ETH-USD Order Book.

---

## Deposit/Withdraw Fees

Coinbase Pro does not charge any additional deposit or withdraw fees for moving funds between your Coinbase accounts and your Exchange accounts.

---

# Data Centers

---

Coinbase Pro data centers are in the Amazon US East N. Virginia (`us-east-1`) region.

---

# Sandbox

---

A public sandbox is available for testing API connectivity and web trading. The sandbox provides all of the functionality of the production exchange but allows you to add fake funds for testing.

Login sessions and API keys are separate from production. Use the sandbox web interface to create keys in the sandbox environment.

To add funds, use the web interface `deposit` and `withdraw` buttons as you would on the production web interface.

---

## Sandbox URLs

When testing your API connectivity, make sure to use the following URLs.

**Website**
`https://public.sandbox.pro.coinbase.com`

**REST API**
`https://api-public.sandbox.pro.coinbase.com`

**Websocket Feed**
`wss://ws-feed-public.sandbox.pro.coinbase.com`

**FIX API**
`tcp+ssl://fix-public.sandbox.pro.coinbase.com:4198`

# Client Libraries

Client libraries can help you integrate with our API quickly.

**OFFICIAL**

- Node.js client library
- GDAX trading toolkit (Node.js)
- Ruby (not actively maintained)

**UNOFFICIAL**

- Go
- Haskell
- Java
- Python
- Rust
- C#

# API

The REST API has endpoints for account and order management as well as public market data.

**REST API ENDPOINT URL**

`https://api.pro.coinbase.com`

There is also a FIX API for order management.

# Requests

All requests and responses are `application/json` content type and follow typical HTTP response status codes for success and failure.

## Errors

```
{
    "message": "Invalid Price"
}
```

Unless otherwise stated, errors to bad requests will respond with HTTP 4xx or status codes. The body will also contain a `message` parameter indicating the cause. Your language's http library should be configured to provide message bodies for non-2xx requests so that you can read the message field from the body.

Common error codes

| Status Code | Reason |
|---|---|
| 400 | Bad Request – Invalid request format |
| 401 | Unauthorized – Invalid API Key |

| Status Code | Reason |
|---|---|
| 403 | Forbidden – You do not have access to the requested resource |
| 404 | Not Found |
| 500 | Internal Server Error – We had a problem with our server |

## Success

A successful response is indicated by HTTP status code 200 and may contain an optional body. If the response has a body it will be documented under each resource below.

# Pagination

Coinbase Pro uses cursor pagination for all REST requests which return arrays. Cursor pagination allows for fetching results before and after the current page of results and is well suited for realtime data. Endpoints like /trades, /fills, /orders, return the latest items by default. To retrieve more results subsequent requests should specify which direction to paginate based on the data previously returned.

`before` and `after` cursors are available via response headers `CB-BEFORE` and `CB-AFTER`. Your requests should use these cursor values when making requests for pages after the initial request.

**PARAMETERS**

| Parameter | Default | Description |
|---|---|---|
| before | | Request page before (newer) this pagination id. |
| after | | Request page after (older) this pagination id. |
| limit | 100 | Number of results per request. Maximum 100. (default 100) |

**EXAMPLE**

```
GET /orders?before=2&limit=30
```

**BEFORE AND AFTER CURSORS**

The `before` cursor references the first item in a results page and the `after` cursor references the last item in a set of results.

To request a page of records before the current one, use the `before` query parameter. Your initial request can omit this parameter to get the default first page.

The response will contain a `CB-BEFORE` header which will return the cursor id to use in your next request for the page before the current one. The page before is a newer page and not one that happened before in chronological time.

The response will also contain a `CB-AFTER` header which will return the cursor id to use in your next request for the page after this one. The page after is an older page and not one that happened after this one in chronological time.

ℹ Cursor pagination can be unintuitive at first. `before` and `after` cursor arguments should not be confused with before and after in chronological time. Most paginated requests return the latest information (newest) as the first page sorted by newest (in chronological time) first. To get older information you would request pages `after` the initial page. To get information newer, you would request pages `before` the first page.

# Types

## Timestamps

```
2014-11-06T10:34:47.123456Z
```

Unless otherwise specified, all timestamps from API are returned in ISO 8601 with microseconds. Make sure you can parse the following ISO 8601 format. Most modern languages and libraries will handle this without issues.

## Numbers

Decimal numbers are returned as strings to preserve full precision across platforms. When making a request, it is recommended that you also convert your numbers to strings to avoid truncation and precision errors.

Integer numbers (like trade id and sequence) are unquoted.

## IDs

Most identifiers are UUID unless otherwise specified. When making a request which requires a UUID, both forms (with and without dashes) are accepted.

`132fb6ae-456b-4654-b4e0-d681ac05cea1` or `132fb6ae456b4654b4e0d681ac05cea1`

# Rate Limits

When a rate limit is exceeded, a status of `429 Too Many Requests` will be returned.

#### REST API

#### PUBLIC ENDPOINTS

We throttle public endpoints by IP: 3 requests per second, up to 6 requests per second in bursts.

#### PRIVATE ENDPOINTS

We throttle private endpoints by user ID: 5 requests per second, up to 10 requests per second in bursts.

#### FINANCIAL INFORMATION EXCHANGE API

The FIX API throttles the number of incoming messages to 50 commands per second.

# Private

Private endpoints are available for order management, and account management. Every private request must be signed using the described authentication scheme.

Private endpoints require authentication using your Coinbase Pro API key. You can generate API keys here

# Authentication

## Generating an API Key

Before being able to sign any requests, you must create an API key via the Coinbase Pro website. Upon creating a key you will have 3 pieces of information which you must remember:

- Key
- Secret
- Passphrase

The Key and Secret will be randomly generated and provided by Coinbase Pro; the Passphrase will be provided by you to further secure your API access. Coinbase Pro stores the salted hash of your passphrase for verification, but cannot recover the passphrase if you forget it.

## API Key Permissions

You can restrict the functionality of API keys. Before creating the key, you must choose what permissions you would like the key to have. The permissions are:

- View - Allows a key read permissions. This includes all GET endpoints.
- Transfer - Allows a key to transfer currency on behalf of an account, including deposits and withdraws. Enable with caution - API key transfers WILL BYPASS two-factor authentication.
- Trade - Allows a key to enter orders, as well as retrieve trade data. This includes POST /orders and several GET endpoints.

Please refer to documentation below to see what API key permissions are required for a specific route.

## Creating a Request

All REST requests must contain the following headers:

- `CB-ACCESS-KEY` The api key as a string.
- `CB-ACCESS-SIGN` The base64-encoded signature (see Signing a Message).
- `CB-ACCESS-TIMESTAMP` A timestamp for your request.
- `CB-ACCESS-PASSPHRASE` The passphrase you specified when creating the API key.

All request bodies should have content type `application/json` and be valid JSON.

## Signing a Message

```
var crypto = require('crypto');
```

```
var secret = 'PYPd1Hv4J6/7x...';

var timestamp = Date.now() / 1000;
var requestPath = '/orders';

var body = JSON.stringify({
    price: '1.0',
    size: '1.0',
    side: 'buy',
    product_id: 'BTC-USD'
});

var method = 'POST';

// create the prehash string by concatenating required parts
var what = timestamp + method + requestPath + body;

// decode the base64 secret
var key = Buffer(secret, 'base64');

// create a sha256 hmac with the secret
var hmac = crypto.createHmac('sha256', key);

// sign the require message with the hmac
// and finally base64 encode the result
return hmac.update(what).digest('base64');
```

The `CB-ACCESS-SIGN` header is generated by creating a sha256 HMAC using the base64-decoded secret key on the prehash string `timestamp + method + requestPath + body` (where `+` represents string concatenation) and base64-encode the output. The timestamp value is the same as the `CB-ACCESS-TIMESTAMP` header.

The `body` is the request body string or omitted if there is no request body (typically for GET requests).

The `method` should be UPPER CASE.

ℹ Remember to first base64-decode the alphanumeric secret string (resulting in 64 bytes) before using it as the key for HMAC. Also, base64-encode the digest output before sending in the header.

## Selecting a Timestamp

The `CB-ACCESS-TIMESTAMP` header MUST be number of seconds since Unix Epoch in UTC. Decimal values are allowed.

Your timestamp must be within 30 seconds of the api service time or your request will be considered expired and rejected. We recommend using the time endpoint to query for the API server time if you believe there many be time skew between your server and the API servers.

# Accounts

## List Accounts

```
[
    {
        "id": "71452118-efc7-4cc4-8780-a5e22d4baa53",
        "currency": "BTC",
        "balance": "0.0000000000000000",
        "available": "0.0000000000000000",
```

```
        "hold": "0.0000000000000000",
        "profile_id": "75da88c5-05bf-4f54-bc85-5c775bd68254"
    },
    {
        "id": "e316cb9a-0808-4fd7-8914-97829c1925de",
        "currency": "USD",
        "balance": "80.2301373066930000",
        "available": "79.2266348066930000",
        "hold": "1.0035025000000000",
        "profile_id": "75da88c5-05bf-4f54-bc85-5c775bd68254"
    }
]
```

Get a list of trading accounts.

Your trading accounts are separate from your Coinbase accounts. See the Deposits section for documentation on how to deposit funds to begin trading.

#### HTTP REQUEST

`GET /accounts`

#### API KEY PERMISSIONS

This endpoint requires either the “view” or “trade” permission.

#### ACCOUNT FIELDS

| Field | Description |
| --- | --- |
| id | Account ID |
| currency | the currency of the account |
| balance | total funds in the account |
| holds | funds on hold (not available for use) |
| available | funds available to withdraw or trade |

#### FUNDS ON HOLD

When you place an order, the funds for the order are placed on hold. They cannot be used for other orders or withdrawn. Funds will remain on hold until the order is filled or canceled.

---

# Get an Account

```
{
    "id": "a1b2c3d4",
    "balance": "1.100",
    "holds": "0.100",
    "available": "1.00",
    "currency": "USD"
}
```

Information for a single account. Use this endpoint when you know the account_id.

#### HTTP REQUEST

`GET /accounts/<account-id>`

#### API KEY PERMISSIONS

This endpoint requires either the “view” or “trade” permission.

#### ACCOUNT FIELDS

| Field | Description |
| --- | --- |
| id | Account ID |
| balance | total funds in the account |
| holds | funds on hold (not available for use) |
| available | funds available to withdraw or trade |

---

# Get Account History


```
[
    {
        "id": "100",
        "created_at": "2014-11-07T08:19:27.028459Z",
        "amount": "0.001",
        "balance": "239.669",
        "type": "fee",
        "details": {
            "order_id": "d50ec984-77a8-460a-b958-66f114b0de9b",
            "trade_id": "74",
            "product_id": "BTC-USD"
        }
    }
]
```


List account activity. Account activity either increases or decreases your account balance. Items are paginated and sorted latest first. See the Pagination section for retrieving additional entries after the first page.

#### HTTP REQUEST

`GET /accounts/<account-id>/ledger`

#### API KEY PERMISSIONS

This endpoint requires either the “view” or “trade” permission.

#### ENTRY TYPES

Entry type indicates the reason for the account change.

| Type | Description |
| --- | --- |
| transfer | Funds moved to/from Coinbase to Coinbase Pro |
| match | Funds moved as a result of a trade |
| fee | Fee as a result of a trade |
| rebate | Fee rebate as per our fee schedule |
| conversion | Funds converted between fiat currency and a stablecoin |

#### DETAILS

If an entry is the result of a trade (match, fee), the `details` field will contain additional information about the trade.

This request is paginated

## Get Holds


```
[
    {
        "id": "82dcd140-c3c7-4507-8de4-2c529cd1a28f",
        "account_id": "e0b3f39a-183d-453e-b754-0c13e5bab0b3",
        "created_at": "2014-11-06T10:34:47.123456Z",
        "updated_at": "2014-11-06T10:40:47.123456Z",
        "amount": "4.23",
        "type": "order",
        "ref": "0a205de4-dd35-4370-a285-fe8fc375a273",
    }
]
```


Holds are placed on an account for any active orders or pending withdraw requests. As an order is filled, the hold amount is updated. If an order is canceled, any remaining hold is removed. For a withdraw, once it is completed, the hold is removed.

#### HTTP REQUEST

`GET /accounts/<account_id>/holds`

#### API KEY PERMISSIONS

This endpoint requires either the “view” or “trade” permission.

This request is paginated

#### TYPE

The type of the hold will indicate why the hold exists. The hold type is `order` for holds related to open orders and `transfer` for holds related to a withdraw.

#### REF

The `ref` field contains the id of the order or transfer which created the hold.

# Orders

## Place a New Order


```
{
    "size": "0.01",
    "price": "0.100",
    "side": "buy",
    "product_id": "BTC-USD"
}
```


Response


```
{
    "id": "d0c5340b-6d6c-49d9-b567-48c4bfca13d2",
```

```
    "price": "0.10000000",
    "size": "0.01000000",
    "product_id": "BTC-USD",
    "side": "buy",
    "stp": "dc",
    "type": "limit",
    "time_in_force": "GTC",
    "post_only": false,
    "created_at": "2016-12-08T20:02:28.53864Z",
    "fill_fees": "0.0000000000000000",
    "filled_size": "0.00000000",
    "executed_value": "0.0000000000000000",
    "status": "pending",
    "settled": false
}
```


You can place two types of orders: `limit` and `market`. Orders can only be placed if your account has sufficient funds. Once an order is placed, your account funds will be put on hold for the duration of the order. How much and which funds are put on hold depends on the order type and parameters specified. See the `Holds` details below.

#### HTTP REQUEST

`POST /orders`

#### API KEY PERMISSIONS

This endpoint requires the “trade” permission.

#### PARAMETERS

These parameters are common to all order types. Depending on the order type, additional parameters will be required (see below).

| Param | Description |
| --- | --- |
| client_oid | [optional] Order ID selected by you to identify your order |
| type | [optional] `limit` or `market` (default is `limit`) |
| side | `buy` or `sell` |
| product_id | A valid product id |
| stp | [optional] Self-trade prevention flag |
| stop | [optional] Either `loss` or `entry`. Requires `stop_price` to be defined. |
| stop_price | [optional] Only if `stop` is defined. Sets trigger price for stop order. |

#### LIMIT ORDER PARAMETERS

| Param | Description |
| --- | --- |
| price | Price per bitcoin |
| size | Amount of BTC to buy or sell |
| time_in_force | [optional] `GTC`, `GTT`, `IOC`, or `FOK` (default is `GTC`) |
| cancel_after | [optional]* `min`, `hour`, `day` |
| post_only | [optional]** Post only flag |

* Requires `time_in_force` to be `GTT`

** Invalid when `time_in_force` is `IOC` or `FOK`

#### MARKET ORDER PARAMETERS

| Param | Description |
| --- | --- |
| size | [optional]* Desired amount in BTC |
| funds | [optional]* Desired amount of quote currency to use |

* One of `size` or `funds` is required.

#### PRODUCT ID

The `product_id` must match a valid product. The products list is available via the /products endpoint.

#### CLIENT ORDER ID

The optional `client_oid` field must be a UUID generated by your trading application. This field value will be broadcast in the public feed for `received` messages. You can use this field to identify your orders in the public feed.

The `client_oid` is different than the server-assigned order id. If you are consuming the public feed and see a `received` message with your `client_oid`, you should record the server-assigned `order_id` as it will be used for future order status updates. The `client_oid` will NOT be used after the `received` message is sent.

The server-assigned order id is also returned as the `id` field to this HTTP POST request.

#### TYPE

When placing an order, you can specify the order type. The order type you specify will influence which other order parameters are required as well as how your order will be executed by the matching engine. If `type` is not specified, the order will default to a `limit` order.

**limit** orders are both the default and basic order type. A limit order requires specifying a `price` and `size`. The `size` is the number of bitcoin to buy or sell, and the `price` is the price per bitcoin. The limit order will be filled at the price specified or better. A sell order can be filled at the specified price per bitcoin or a higher price per bitcoin and a buy order can be filled at the specified price or a lower price depending on market conditions. If market conditions cannot fill the limit order immediately, then the limit order will become part of the open order book until filled by another incoming order or canceled by the user.

**market** orders differ from limit orders in that they provide no pricing guarantees. They however do provide a way to buy or sell specific amounts of bitcoin or fiat without having to specify the price. Market orders execute immediately and no part of the market order will go on the open order book. Market orders are always considered `takers` and incur taker fees. When placing a market order you can specify `funds` and/or `size`. Funds will limit how much of your quote currency account balance is used and size will limit the bitcoin amount transacted.

#### STOP ORDERS

Stop orders become active and wait to trigger based on the movement of the last trade price. There are two types of stop orders, `stop loss` and `stop entry`:

`stop: 'loss'`: Triggers when the last trade price changes to a value at or below the `stop_price`.

`stop: 'entry'`: Triggers when the last trade price changes to a value at or above the `stop_price`.

The last trade price is the last price at which an order was filled. This price can be found in the latest match message. Note that not all match messages may be received due to dropped messages.

Note that when triggered, stop orders execute as either market or limit orders, depending on the `type`. They are therefore subject to holds.

#### PRICE

The price must be specified in `quote_increment` product units. The quote increment is the smallest unit of price. For the BTC-USD product, the quote increment is `0.01` or 1 penny. Prices less than 1 penny will not be accepted, and no fractional penny prices will be accepted. Not required for `market` orders.

#### SIZE

The size must be greater than the `base_min_size` for the product and no larger than the `base_max_size`. The size can be in any increment of the base currency (BTC for the BTC-USD product), which includes satoshi units. `size` indicates the amount of BTC (or

base currency) to buy or sell.

#### FUNDS

The funds field is optionally used for `market` orders. When specified it indicates how much of the product quote currency to buy or sell. For example, a market buy for `BTC-USD` with `funds` specified as `150.00` will spend `150 USD` to buy BTC (including any fees). If the funds field is not specified for a market buy order, `size` must be specified and Coinbase Pro will use available funds in your account to buy bitcoin.

A market sell order can also specify the `funds`. If `funds` is specified, it will limit the sell to the amount of `funds` specified. You can use `funds` with sell orders to limit the amount of quote currency funds received.

#### TIME IN FORCE

Time in force policies provide guarantees about the lifetime of an order. There are four policies: good till canceled `GTC`, good till time `GTT`, immediate or cancel `IOC`, and fill or kill `FOK`.

`GTC` Good till canceled orders remain open on the book until canceled. This is the default behavior if no policy is specified.

`GTT` Good till time orders remain open on the book until canceled or the allotted `cancel_after` is depleted on the matching engine. `GTT` orders are guaranteed to cancel before any other order is processed after the `cancel_after` timestamp which is returned by the API. A `day` is considered 24 hours.

`IOC` Immediate or cancel orders instantly cancel the remaining size of the limit order instead of opening it on the book.

`FOK` Fill or kill orders are rejected if the entire size cannot be matched.

* Note, match also refers to self trades.

#### POST ONLY

The post-only flag indicates that the order should only make liquidity. If any part of the order results in taking liquidity, the order will be rejected and no part of it will execute.

#### HOLDS

For `limit` `buy` orders, we will hold `price x size x (1 + fee-percent)` USD. For `sell` orders, we will hold the number of Bitcoin you wish to sell. Actual fees are assessed at time of trade. If you cancel a partially filled or unfilled order, any remaining funds will be released from hold.

For `market` `buy` orders where `funds` is specified, the `funds` amount will be put on hold. If only `size` is specified, all of your account balance (in the quote account) will be put on hold for the duration of the market order (usually a trivially short time). For a `sell` order, the `size` in BTC will be put on hold. If `size` is not specified (and only `funds` is specified), your entire BTC balance will be on hold for the duration of the market order.

#### SELF-TRADE PREVENTION

Self-trading is not allowed on Coinbase Pro. Two orders from the same user will not be allowed to match with one another. To change the self-trade behavior, specify the `stp` flag.

| Flag | Name |
|---|---|
| dc | Decrease and Cancel (default) |
| co | Cancel oldest |
| cn | Cancel newest |
| cb | Cancel both |

See the self-trade prevention documentation for details about these fields.

#### ORDER LIFECYCLE

The HTTP Request will respond when an order is either rejected (insufficient funds, invalid parameters, etc) or received (accepted by the matching engine). A `200` response indicates that the order was received and is active. Active orders may execute immediately

(depending on price and market conditions) either partially or fully. A partial execution will put the remaining size of the order in the `open` state. An order that is filled completely, will go into the `done` state.

Users listening to streaming market data are encouraged to use the `client_oid` field to identify their `received` messages in the feed. The REST response with a server `order_id` may come after the `received` message in the public data feed.

#### RESPONSE

A successful order will be assigned an order id. A successful order is defined as one that has been accepted by the matching engine.

> Open orders do not expire and will remain open until they are either filled or canceled.

---

# Cancel an Order

Cancel a previously placed order.

If the order had no matches during its lifetime its record may be purged. This means the order details will not be available with `GET /orders/<order-id>`.

#### HTTP REQUEST

`DELETE /orders/<order-id>`

#### API KEY PERMISSIONS

This endpoint requires the “trade” permission.

> The `order id` is the server-assigned order id and not the optional client_oid.

#### CANCEL REJECT

If the order could not be canceled (already filled or previously canceled, etc), then an error response will indicate the reason in the `message` field.

---

# Cancel all

With best effort, cancel all open orders. The response is a list of ids of the canceled orders.

```
[
    "144c6f8e-713f-4682-8435-5280fbe8b2b4",
    "debe4907-95dc-442f-af3b-cec12f42ebda",
    "cf7aceee-7b08-4227-a76c-3858144323ab",
    "dfc5ae27-cadb-4c0c-beef-8994936fde8a",
    "34fecfbf-de33-4273-b2c6-baf8e8948be4"
]
```

#### HTTP REQUEST

`DELETE /orders`

#### API KEY PERMISSIONS

This endpoint requires the “trade” permission.

#### QUERY PARAMETERS

| Param | Default | Description |
| --- | --- | --- |
| product_id | [optional] | Only cancel orders open for a specific product |

# List Orders


```
[
    {
        "id": "d0c5340b-6d6c-49d9-b567-48c4bfca13d2",
        "price": "0.10000000",
        "size": "0.01000000",
        "product_id": "BTC-USD",
        "side": "buy",
        "stp": "dc",
        "type": "limit",
        "time_in_force": "GTC",
        "post_only": false,
        "created_at": "2016-12-08T20:02:28.53864Z",
        "fill_fees": "0.0000000000000000",
        "filled_size": "0.00000000",
        "executed_value": "0.0000000000000000",
        "status": "open",
        "settled": false
    },
    {
        "id": "8b99b139-58f2-4ab2-8e7a-c11c846e3022",
        "price": "1.00000000",
        "size": "1.00000000",
        "product_id": "BTC-USD",
        "side": "buy",
        "stp": "dc",
        "type": "limit",
        "time_in_force": "GTC",
        "post_only": false,
        "created_at": "2016-12-08T20:01:19.038644Z",
        "fill_fees": "0.0000000000000000",
        "filled_size": "0.00000000",
        "executed_value": "0.0000000000000000",
        "status": "open",
        "settled": false
    }
]
```


List your current open orders. Only open or un-settled orders are returned. As soon as an order is no longer open and settled, it will no longer appear in the default request.

#### HTTP REQUEST

`GET /orders`

#### API KEY PERMISSIONS

This endpoint requires either the “view” or “trade” permission.

#### QUERY PARAMETERS

| Param | Default | Description |
| --- | --- | --- |
| status | [open, pending, active] | Limit list of orders to these statuses. Passing `all` returns orders of all statuses. |
| product_id | [optional] | Only list orders for a specific product |

To specify multiple statuses, use the `status` query argument multiple times: `/orders?status=done&status=pending`.

This request is paginated.

#### ORDER STATUS AND SETTLEMENT

Orders which are no longer resting on the order book, will be marked with the `done` status. There is a small window between an order being `done` and `settled`. An order is settled when all of the fills have settled and the remaining holds (if any) have been removed.

#### POLLING

For high-volume trading it is strongly recommended that you maintain your own list of open orders and use one of the streaming market data feeds to keep it updated. You should poll the open orders endpoint once when you start trading to obtain the current state of any open orders.

`executed_value` is the cumulative match size * price and is only present for orders placed after 2016-05-20.

Open orders may change state between the request and the response depending on market conditions.

---

# Get an Order


```
{
    "id": "68e6a28f-ae28-4788-8d4f-5ab4e5e5ae08",
    "size": "1.00000000",
    "product_id": "BTC-USD",
    "side": "buy",
    "stp": "dc",
    "funds": "9.9750623400000000",
    "specified_funds": "10.0000000000000000",
    "type": "market",
    "post_only": false,
    "created_at": "2016-12-08T20:09:05.508883Z",
    "done_at": "2016-12-08T20:09:05.527Z",
    "done_reason": "filled",
    "fill_fees": "0.0249376391550000",
    "filled_size": "0.01291771",
    "executed_value": "9.9750556620000000",
    "status": "done",
    "settled": true
}
```


Get a single order by order id.

#### HTTP REQUEST

`GET /orders/<order-id>`

#### API KEY PERMISSIONS

This endpoint requires either the “view” or “trade” permission.

If the order is canceled the response may have status code `404` if the order had no matches.

Open orders may change state between the request and the response depending on market conditions.

---

# Fills

---

# List Fills

```
[
    {
        "trade_id": 74,
        "product_id": "BTC-USD",
        "price": "10.00",
        "size": "0.01",
        "order_id": "d50ec984-77a8-460a-b958-66f114b0de9b",
        "created_at": "2014-11-07T22:19:28.578544Z",
        "liquidity": "T",
        "fee": "0.00025",
        "settled": true,
        "side": "buy"
    }
]
```

Get a list of recent fills.

#### HTTP REQUEST

`GET /fills`

#### API KEY PERMISSIONS

This endpoint requires either the “view” or “trade” permission.

#### QUERY PARAMETERS

You can request fills for specific orders or products using query parameters.

| Param | Default | Description |
|---|---|---|
| order_id | *all* | Limit list of fills to this order_id |
| product_id | *all* | Limit list of fills to this product_id |

`DEPRECATION NOTICE` - Requests without either `order_id` or `product_id` will be rejected after 8/23/18.

#### SETTLEMENT AND FEES

Fees are recorded in two stages. Immediately after the matching engine completes a match, the fill is inserted into our datastore. Once the fill is recorded, a settlement process will settle the fill and credit both trading counterparties.

The `fee` field indicates the fees charged for this individual fill.

#### LIQUIDITY

The `liquidity` field indicates if the fill was the result of a liquidity provider or liquidity taker. `M` indicates Maker and `T` indicates Taker.

#### PAGINATION

Fills are returned sorted by descending `trade_id` from the largest `trade_id` to the smallest `trade_id`. The `CB-BEFORE` header will have this first trade id so that future requests using the `cb-before` parameter will fetch fills with a greater trade id (newer fills).

This request is paginated.

# Deposits

# Payment method

Request

```
{
    "amount": 10.00,
    "currency": "USD",
    "payment_method_id": "bc677162-d934-5f1a-968c-a496b1c1270b"
}
```

Response

```
{
    "id": "593533d2-ff31-46e0-b22e-ca754147a96a",
    "amount": "10.00",
    "currency": "USD",
    "payout_at": "2016-08-20T00:31:09Z"
}
```

Deposit funds from a payment method. See the Payment Methods section for retrieving your payment methods.

#### HTTP REQUEST

`POST /deposits/payment-method`

#### API KEY PERMISSIONS

This endpoint requires the “transfer” permission.

#### PARAMETERS

| Param | Description |
| --- | --- |
| amount | The amount to deposit |
| currency | The type of currency |
| payment_method_id | ID of the payment method |

---

# Coinbase

Request

```
{
    "amount": 10.00,
    "currency": "BTC",
    "coinbase_account_id": "c13cd0fc-72ca-55e9-843b-b84ef628c198",
}
```

Response

```
{
    "id": "593533d2-ff31-46e0-b22e-ca754147a96a",
    "amount": "10.00",
    "currency": "BTC",
}
```

Deposit funds from a coinbase account. You can move funds between your Coinbase accounts and your Coinbase Pro trading accounts within your daily limits. Moving funds between Coinbase and Coinbase Pro is instant and free. See the Coinbase Accounts

section for retrieving your Coinbase accounts.

#### HTTP REQUEST

`POST /deposits/coinbase-account`

#### API KEY PERMISSIONS

This endpoint requires the “transfer” permission.

#### PARAMETERS

| Param | Description |
| --- | --- |
| amount | The amount to deposit |
| currency | The type of currency |
| coinbase_account_id | ID of the coinbase account |

# Withdrawals

## Payment method

Request

```
{
    "amount": 10.00,
    "currency": "USD",
    "payment_method_id": "bc677162-d934-5f1a-968c-a496b1c1270b"
}
```

Response

```
{
    "id":"593533d2-ff31-46e0-b22e-ca754147a96a",
    "amount": "10.00",
    "currency": "USD",
    "payout_at": "2016-08-20T00:31:09Z"
}
```

Withdraw funds to a payment method. See the Payment Methods section for retrieving your payment methods.

#### HTTP REQUEST

`POST /withdrawals/payment-method`

#### API KEY PERMISSIONS

This endpoint requires the “transfer” permission.

#### PARAMETERS

| Param | Description |
| --- | --- |
| amount | The amount to withdraw |
| currency | The type of currency |
| payment_method_id | ID of the payment method |

# Coinbase

Request

```
{
    "amount": 10.00,
    "currency": "BTC",
    "coinbase_account_id": "c13cd0fc-72ca-55e9-843b-b84ef628c198",
}
```

Response

```
{
    "id":"593533d2-ff31-46e0-b22e-ca754147a96a",
    "amount":"10.00",
    "currency": "BTC",
}
```

Withdraw funds to a coinbase account. You can move funds between your Coinbase accounts and your Coinbase Pro trading accounts within your daily limits. Moving funds between Coinbase and Coinbase Pro is instant and free. See the Coinbase Accounts section for retrieving your Coinbase accounts.

#### HTTP REQUEST

`POST /withdrawals/coinbase-account`

#### API KEY PERMISSIONS

This endpoint requires the “transfer” permission.

#### PARAMETERS

| Param | Description |
| --- | --- |
| amount | The amount to withdraw |
| currency | The type of currency |
| coinbase_account_id | ID of the coinbase account |

# Crypto

Request

```
{
    "amount": 10.00,
    "currency": "BTC",
    "crypto_address": "0x5ad5769cd04681FeD900BCE3DDc877B50E83d469"
}
```

Response

```
{
    "id":"593533d2-ff31-46e0-b22e-ca754147a96a",
    "amount":"10.00",
    "currency": "BTC",
}
```

Withdraws funds to a crypto address.

#### HTTP REQUEST

`POST /withdrawals/crypto`

#### API KEY PERMISSIONS

This endpoint requires the “transfer” permission.

#### PARAMETERS

| Param | Description |
| --- | --- |
| amount | The amount to withdraw |
| currency | The type of currency |
| crypto_address | A crypto address of the recipient |

# Stablecoin Conversions

## Create Conversion

Request

```
{
    "from": "USD",
    "to": "USDC",
    "amount": "10000.00"
}
```

Response

```
{
    "id": "8942caee-f9d5-4600-a894-4811268545db",
    "amount": "10000.00",
    "from_account_id": "7849cc79-8b01-4793-9345-bc6b5f08acce",
    "to_account_id": "105c3e58-0898-4106-8283-dc5781cda07b",
    "from": "USD",
    "to": "USDC"
}
```

Convert $10,000.00 to 10,000.00 USDC.

#### HTTP REQUEST

`POST /conversions`

#### API KEY PERMISSIONS

This endpoint requires the “trade” permission.

#### PARAMETERS

| Param | Description |
| --- | --- |
| from | A valid currency id |
| to | A valid currency id |

| Param | Description |
| --- | --- |
| amount | Amount of `from` to convert to `to` |

**RESPONSE**

A successful conversion will be assigned a conversion id. The corresponding ledger entries for a conversion will reference this conversion id.

# Payment Methods

## List Payment Methods


```
[
    {
        "id": "bc6d7162-d984-5ffa-963c-a493b1c1370b",
        "type": "ach_bank_account",
        "name": "Bank of America - eBan... ********7134",
        "currency": "USD",
        "primary_buy": true,
        "primary_sell": true,
        "allow_buy": true,
        "allow_sell": true,
        "allow_deposit": true,
        "allow_withdraw": true,
        "limits": {
            "buy": [
                {
                    "period_in_days": 1,
                    "total": {
                        "amount": "10000.00",
                        "currency": "USD"
                    },
                    "remaining": {
                        "amount": "10000.00",
                        "currency": "USD"
                    }
                }
            ],
            "instant_buy": [
                {
                    "period_in_days": 7,
                    "total": {
                        "amount": "0.00",
                        "currency": "USD"
                    },
                    "remaining": {
                        "amount": "0.00",
                        "currency": "USD"
                    }
                }
            ],
            "sell": [
                {
                    "period_in_days": 1,
                    "total": {
                        "amount": "10000.00",
                        "currency": "USD"
                    },
                    "remaining": {
                        "amount": "10000.00",
                        "currency": "USD"
                    }
                }
            ],
```

```
            "deposit": [
                {
                    "period_in_days": 1,
                    "total": {
                        "amount": "10000.00",
                        "currency": "USD"
                    },
                    "remaining": {
                        "amount": "10000.00",
                        "currency": "USD"
                    }
                }
            ]
        }
    },
]
```


Get a list of your payment methods.

#### HTTP REQUEST

`GET /payment-methods`

#### API KEY PERMISSIONS

This endpoint requires the “transfer” permission.

# Coinbase Accounts

## List Accounts


```
[
    {
        "id": "fc3a8a57-7142-542d-8436-95a3d82e1622",
        "name": "ETH Wallet",
        "balance": "0.00000000",
        "currency": "ETH",
        "type": "wallet",
        "primary": false,
        "active": true
    },
    {
        "id": "2ae3354e-f1c3-5771-8a37-6228e9d239db",
        "name": "USD Wallet",
        "balance": "0.00",
        "currency": "USD",
        "type": "fiat",
        "primary": false,
        "active": true,
        "wire_deposit_information": {
            "account_number": "0199003122",
            "routing_number": "026013356",
            "bank_name": "Metropolitan Commercial Bank",
            "bank_address": "99 Park Ave 4th Fl New York, NY 10016",
            "bank_country": {
                "code": "US",
                "name": "United States"
            },
            "account_name": "Coinbase, Inc",
            "account_address": "548 Market Street, #23008, San Francisco, CA 94104",
            "reference": "BAOCAEUX"
        }
    },
    {
        "id": "1bfad868-5223-5d3c-8a22-b5ed371e55cb",
        "name": "BTC Wallet",
```

```
        "balance": "0.00000000",
        "currency": "BTC",
        "type": "wallet",
        "primary": true,
        "active": true
    },
    {
        "id": "2a11354e-f133-5771-8a37-622be9b239db",
        "name": "EUR Wallet",
        "balance": "0.00",
        "currency": "EUR",
        "type": "fiat",
        "primary": false,
        "active": true,
        "sepa_deposit_information": {
            "iban": "EE957700771001355096",
            "swift": "LHVBEE22",
            "bank_name": "AS LHV Pank",
            "bank_address": "Tartu mnt 2, 10145 Tallinn, Estonia",
            "bank_country_name": "Estonia",
            "account_name": "Coinbase UK, Ltd.",
            "account_address": "9th Floor, 107 Cheapside, London, EC2V 6DN, United Kingdom",
            "reference": "CBAEUXOVFXOXYX"
        }
    },
]
```


Get a list of your coinbase accounts.

Visit the Coinbase accounts API for more information.

#### HTTP REQUEST

`GET /coinbase-accounts`

#### API KEY PERMISSIONS

This endpoint requires either the “view” or “transfer” permission.

# Reports

## Create a new report

Request


```
{
    "type": "fills",
    "start_date": "2014-11-01T00:00:00.000Z",
    "end_date": "2014-11-30T23:59:59.000Z"
}
```


Response


```
{
    "id": "0428b97b-bec1-429e-a94c-59232926778d",
    "type": "fills",
    "status": "pending",
    "created_at": "2015-01-06T10:34:47.000Z",
    "completed_at": undefined,
    "expires_at": "2015-01-13T10:35:47.000Z",
    "file_url": undefined,
    "params": {
        "start_date": "2014-11-01T00:00:00.000Z",
```

```
        "end_date": "2014-11-30T23:59:59.000Z"
    }
}
```

Reports provide batches of historic information about your account in various human and machine readable forms.

#### HTTP REQUEST

`POST /reports`

#### API KEY PERMISSIONS

This endpoint requires either the “view” or “trade” permission.

#### PARAMETERS

| Param | Description |
| --- | --- |
| type | `fills` or `account` |
| start_date | Starting date for the report (inclusive) |
| end_date | Ending date for the report (inclusive) |
| product_id | ID of the product to generate a fills report for. E.g. BTC-USD. Required if `type` is `fills` |
| account_id | ID of the account to generate an account report for. Required if `type` is `account` |
| format | `pdf` or `csv` (defualt is `pdf`) |
| email | Email address to send the report to (optional) |

The report will be generated when resources are available. Report status can be queried via the `/reports/:report_id` endpoint. The `file_url` field will be available once the report has successfully been created and is available for download.

#### EXPIRED REPORTS

Reports are only available for download for a few days after being created. Once a report expires, the report is no longer available for download and is deleted.

---

# Get report status

Response (creating report)

```
{
    "id": "0428b97b-bec1-429e-a94c-59232926778d",
    "type": "fills",
    "status": "creating",
    "created_at": "2015-01-06T10:34:47.000Z",
    "completed_at": undefined,
    "expires_at": "2015-01-13T10:35:47.000Z",
    "file_url": undefined,
    "params": {
        "start_date": "2014-11-01T00:00:00.000Z",
        "end_date": "2014-11-30T23:59:59.000Z"
    }
}
```

Response (finished report)

```
{
    "id": "0428b97b-bec1-429e-a94c-59232926778d",
```

```
    "type": "fills",
    "status": "ready",
    "created_at": "2015-01-06T10:34:47.000Z",
    "completed_at": "2015-01-06T10:35:47.000Z",
    "expires_at": "2015-01-13T10:35:47.000Z",
    "file_url": "https://example.com/0428b97b.../fills.pdf",
    "params": {
        "start_date": "2014-11-01T00:00:00.000Z",
        "end_date": "2014-11-30T23:59:59.000Z"
    }
}
```

#### HTTP REQUEST

`GET /reports/:report_id`

Once a report request has been accepted for processing, the status is available by polling the report resource endpoint.

The final report will be uploaded and available at `file_url` once the `status` indicates `ready`

#### API KEY PERMISSIONS

This endpoint requires either the “view” or “trade” permission.

#### STATUS

| Status | Description |
| --- | --- |
| pending | The report request has been accepted and is awaiting processing |
| creating | The report is being created |
| ready | The report is ready for download from `file_url` |

# User Account

## Trailing Volume

```
[
    {
        "product_id": "BTC-USD",
        "exchange_volume": "11800.00000000",
        "volume": "100.00000000",
        "recorded_at": "1973-11-29T00:05:01.123456Z"
    },
    {
        "product_id": "LTC-USD",
        "exchange_volume": "51010.04100000",
        "volume": "2010.04100000",
        "recorded_at": "1973-11-29T00:05:02.123456Z"
    }
]
```

#### HTTP REQUEST

`GET /users/self/trailing-volume`

#### API KEY PERMISSIONS

This endpoint requires either the “view” or “trade” permission.

This request will return your 30-day trailing volume for all products. This is a cached value that’s calculated every day at midnight UTC.

# Market Data

The Market Data API is an unauthenticated set of endpoints for retrieving market data. These endpoints provide snapshots of market data.

> By accessing the Coinbase Pro Market Data API, you agree to be bound by the Market Data Terms of Use.

> For real-time market data updates, see the Websocket Feed documentation for connecting and re-creating a perfect real-time copy of the order book and trades.

# Products

## Get Products

```
[
    {
        "id": "BTC-USD",
        "base_currency": "BTC",
        "quote_currency": "USD",
        "base_min_size": "0.001",
        "base_max_size": "10000.00",
        "quote_increment": "0.01"
    }
]
```

Get a list of available currency pairs for trading.

**HTTP REQUEST**

`GET /products`

**DETAILS**

The `base_min_size` and `base_max_size` fields define the min and max order size. The `quote_increment` field specifies the min order price as well as the price increment.

The order price must be a multiple of this increment (i.e. if the increment is 0.01, order prices of 0.001 or 0.021 would be rejected).

> Product ID will not change once assigned to a product but the min/max/quote sizes can be updated in the future.

## Get Product Order Book

Example Response for **/products/BTC-USD/book** Only the best bid and ask is returned.

```
{
    "sequence": "3",
    "bids": [
```

```
        [ price, size, num-orders ],
    ],
    "asks": [
        [ price, size, num-orders ],
    ]
}
```

Example Response for **/products/BTC-USD/book?level=2**

```
{
    "sequence": "3",
    "bids": [
        [ price, size, num-orders ],
        [ "295.96", "4.39088265", 2 ],
        ...
    ],
    "asks": [
        [ price, size, num-orders ],
        [ "295.97", "25.23542881", 12 ],
        ...
    ]
}
```

Example Response for **/products/BTC-USD/book?level=3**

```
{
    "sequence": "3",
    "bids": [
        [ price, size, order_id ],
        [ "295.96","0.05088265","3b0f1225-7f84-490b-a29f-0faef9de823a" ],
        ...
    ],
    "asks": [
        [ price, size, order_id ],
        [ "295.97","5.72036512","da863862-25f4-4868-ac41-005d11ab0a5f" ],
        ...
    ]
}
```

Get a list of open orders for a product. The amount of detail shown can be customized with the `level` parameter.

#### HTTP REQUEST

`GET /products/<product-id>/book`

#### DETAILS

By default, only the inside (i.e. best) bid and ask are returned. This is equivalent to a book depth of 1 level. If you would like to see a larger order book, specify the `level` query parameter.

If a level is not aggregated, then all of the orders at each price will be returned. Aggregated levels return only one size for each active price (as if there was only a single order for that size at the level).

#### PARAMETERS

| Name | Default | Description |
|---|---|---|
| level | 1 | Select response detail. Valid levels are documented below |

#### LEVELS

| Level | Description |
|---|---|
| 1 | Only the best bid and ask |
| 2 | Top 50 bids and asks (aggregated) |

| Level | Description |
| --- | --- |
| 3 | Full order book (non aggregated) |

Levels 1 and 2 are aggregated. The `size` field is the sum of the size of the orders at that `price`, and `num-orders` is the count of orders at that `price`; `size` should not be multiplied by `num-orders`.

Level 3 is non-aggregated and returns the entire order book.

ℹ This request is NOT paginated. The entire book is returned in one response.

ℹ Level 1 and Level 2 are recommended for polling. For the most up-to-date data, consider using the websocket stream.

❗ Level 3 is only recommended for users wishing to maintain a full real-time order book using the websocket stream. Abuse of Level 3 via polling will cause your access to be limited or blocked.

---

# Get Product Ticker


```
{
  "trade_id": 4729088,
  "price": "333.99",
  "size": "0.193",
  "bid": "333.98",
  "ask": "333.99",
  "volume": "5957.11914015",
  "time": "2015-11-14T20:46:03.511254Z"
}
```


Snapshot information about the last trade (tick), best bid/ask and 24h volume.

#### HTTP REQUEST

`GET /products/<product-id>/ticker`

#### REAL-TIME UPDATES

Polling is discouraged in favor of connecting via the websocket stream and listening for `match` messages.

---

# Get Trades


```
[{
    "time": "2014-11-07T22:19:28.578544Z",
    "trade_id": 74,
    "price": "10.00000000",
    "size": "0.01000000",
    "side": "buy"
}, {
    "time": "2014-11-07T01:08:43.642366Z",
    "trade_id": 73,
    "price": "100.00000000",
    "size": "0.01000000",
```

```
    "side": "sell"
}]
```

List the latest trades for a product.

#### HTTP REQUEST

`GET /products/<product-id>/trades`

> This request is paginated.

#### SIDE

The trade `side` indicates the maker order side. The maker order is the order that was open on the order book. `buy` side indicates a down-tick because the maker was a buy order and their order was removed. Conversely, `sell` side indicates an up-tick.

---

# Get Historic Rates

```
[
    [ time, low, high, open, close, volume ],
    [ 1415398768, 0.32, 4.2, 0.35, 4.2, 12.3 ],
    ...
]
```

Historic rates for a product. Rates are returned in grouped buckets based on requested `granularity`.

> Historical rate data may be incomplete. No data is published for intervals where there are no ticks.

> Historical rates should not be polled frequently. If you need real-time information, use the trade and book endpoints along with the websocket feed.

#### HTTP REQUEST

`GET /products/<product-id>/candles`

#### PARAMETERS

| Param | Description |
|---|---|
| start | Start time in ISO 8601 |
| end | End time in ISO 8601 |
| granularity | Desired timeslice in seconds |

#### DETAILS

If either one of the `start` or `end` fields are not provided then both fields will be ignored. If a custom time range is not declared then one ending now is selected.

The `granularity` field must be one of the following values: `{60, 300, 900, 3600, 21600, 86400}`. Otherwise, your request will be rejected. These values correspond to timeslices representing one minute, five minutes, fifteen minutes, one hour, six hours, and one day, respectively.

If data points are readily available, your response may contain as many as 300 candles and some of those candles may precede your declared `start` value.

The maximum number of data points for a single request is 300 candles. If your selection of start/end time and granularity will result in more than 300 data points, your request will be rejected. If you wish to retrieve fine granularity data over a larger time range, you will need to make multiple requests with new start/end ranges.

#### RESPONSE ITEMS

Each bucket is an array of the following information:

- `time` bucket start time
- `low` lowest price during the bucket interval
- `high` highest price during the bucket interval
- `open` opening price (first trade) in the bucket interval
- `close` closing price (last trade) in the bucket interval
- `volume` volume of trading activity during the bucket interval

---

## Get 24hr Stats

```
{
    "open": "34.19000000",
    "high": "95.70000000",
    "low": "7.06000000",
    "volume": "2.41000000"
}
```

Get 24 hr stats for the product. `volume` is in base currency units. `open`, `high`, `low` are in quote currency units.

#### HTTP REQUEST

`GET /products/<product-id>/stats`

---

# Currencies

---

## Get currencies

```
[{
    "id": "BTC",
    "name": "Bitcoin",
    "min_size": "0.00000001"
}, {
    "id": "USD",
    "name": "United States Dollar",
    "min_size": "0.01000000"
}]
```

List known currencies.

#### HTTP REQUEST

`GET /currencies`

Not all currencies may be currently in use for trading.

#### CURRENCY CODES

Currency codes will conform to the ISO 4217 standard where possible. Currencies which have or had no representation in ISO 4217 may use a custom code.

| Code | Description |
| --- | --- |
| BTC | Bitcoin |
| ETH | Ether |
| LTC | Litecoin |

# Time

This endpoint **does not** require authentication.

```
{
    "iso": "2015-01-07T23:47:25.201Z",
    "epoch": 1420674445.201
}
```

Get the API server time.

#### HTTP REQUEST

`GET /time`

#### EPOCH

The `epoch` field represents decimal seconds since Unix Epoch

# Websocket Feed

The websocket feed provides real-time market data updates for orders and trades.

`wss://ws-feed.pro.coinbase.com`

# Overview

Real-time market data updates provide the fastest insight into order flow and trades. This however means that you are responsible for reading the message stream and using the message relevant for your needs which can include building real-time order books or tracking real-time trades.

The websocket feed is publicly available, but connections to it are rate-limited to 1 per 4 seconds per IP.

# Protocol overview

The websocket feed uses a bidirectional protocol, which encodes all messages as JSON objects. All messages have a `type` attribute that can be used to handle the message appropriately.

Please note that new message types can be added at any point in time. Clients are expected to ignore messages they do not support.

**Error messages:** Most failure cases will cause an `error` message (a message with the `type` `"error"`) to be emitted. This can be helpful for implementing a client or debugging issues.

```
{
    "type": "error",
    "message": "error message",
    /* ... */
}
```

# Subscribe

```
// Request
// Subscribe to ETH-USD and ETH-EUR with the level2, heartbeat and ticker channels,
// plus receive the ticker entries for ETH-BTC and ETH-USD
{
    "type": "subscribe",
    "product_ids": [
        "ETH-USD",
        "ETH-EUR"
    ],
    "channels": [
        "level2",
        "heartbeat",
        {
            "name": "ticker",
            "product_ids": [
                "ETH-BTC",
                "ETH-USD"
            ]
        }
    ]
}
```

To begin receiving feed messages, you must first send a `subscribe` message to the server indicating which channels and products to receive. This message is mandatory — you will be disconnected if no `subscribe` has been received within 5 seconds.

There are two ways to specify products ids to listen for within each channel: First, you can specify the product ids for an individual channel. Also, as a shorthand, you can define products ids at the root of the object, which will add them to all the channels you subscribe to.

```
// Response
{
    "type": "subscriptions",
    "channels": [
        {
            "name": "level2",
            "product_ids": [
                "ETH-USD",
                "ETH-EUR"
            ],
        },
        {
            "name": "heartbeat",
```

```
            "product_ids": [
                "ETH-USD",
                "ETH-EUR"
            ],
        },
        {
            "name": "ticker",
            "product_ids": [
                "ETH-USD",
                "ETH-EUR",
                "ETH-BTC"
            ]
        }
    ]
}
```

Once a `subscribe` message is received the server will respond with a `subscriptions` message that lists all channels you are subscribed to.

Subsequent subscribe messages will add to the list of subscriptions. In case you already subscribed to a channel without being authenticated you will remain in the unauthenticated channel.

If you want to unsubscribe from channel/product pairs, send an `unsubscribe` message. The structure is equivalent to `subscribe` messages. As a shorthand you can also provide no product ids for a channel, which will unsubscribe you from the channel entirely.

```
// Request
{
    "type": "unsubscribe",
    "product_ids": [
        "ETH-USD",
        "ETH-EUR"
    ],
    "channels": ["ticker"]
}
```

```
// Request
{
    "type": "unsubscribe",
    "channels": ["heartbeat"]
}
```

As a response to an `unsubscribe` message you will receive a `subscriptions` message.

#### AUTHENTICATION

It is possible to authenticate yourself when subscribing to the websocket feed.

Authentication will result in a couple of benefits:

1. Messages where you're one of the parties are expanded and have more useful fields
2. You will receive private messages, such as lifecycle information about stop orders you placed

```
// Authenticated feed messages add user_id and
// profile_id for messages related to your user
{
    "type": "open", // "received" | "open" | "done" | "match" | "change" | "activate"
    "user_id": "5844eceecf7e803e259d0365",
    "profile_id": "765d1549-9660-4be2-97d4-fa2d65fa3352",
    /* ... */
}
```

Here's an example of an authenticated `subscribe` request:

```
// Request
{
    "type": "subscribe",
```

```
    "product_ids": [
        "BTC-USD"
    ],
    "channels": ["full"],
    "signature": "...",
    "key": "...",
    "passphrase": "...",
    "timestamp": "..."
}
```

To authenticate, you send a `subscribe` message as usual, but you also pass in fields just as if you were signing a request to `GET /users/self/verify`. To get the necessary parameters, you would go through the same process as you do to make authenticated calls to the API.

Authenticated feed messages do not increment the sequence number. It is currently not possible to detect if an authenticated feed message was dropped.

The easiest way to connect to an authenticated feed are our gdax-node and GDAX trading toolkit libraries.

## Sequence Numbers

Most feed messages contain a sequence number. Sequence numbers are increasing integer values for each product with every new message being exactly 1 sequence number than the one before it.

If you see a sequence number that is more than one value from the previous, it means a message has been dropped. A sequence number less than one you have seen can be ignored or has arrived out-of-order. In both situations you may need to perform logic to make sure your system is in the correct state.

While a websocket connection is over TCP, the websocket servers receive market data in a manner which can result in dropped messages. Your feed consumer should either be designed to expect and handle sequence gaps and out-of-order messages, or use channels that guarantee delivery of messages.

If you want to keep an order book in sync, consider using the level 2 channel, which provides such guarantees.

# Channels

## The heartbeat channel

```
// Request
{
    "type": "subscribe",
    "channels": [{ "name": "heartbeat", "product_ids": ["ETH-EUR"] }]
}
```

To receive heartbeat messages for specific products once a second subscribe to the `heartbeat` channel. Heartbeats also include sequence numbers and last trade ids that can be used to verify no messages were missed.

```
// Heartbeat message
{
```

```
    "type": "heartbeat",
    "sequence": 90,
    "last_trade_id": 20,
    "product_id": "BTC-USD",
    "time": "2014-11-07T08:19:28.464459Z"
}
```

# The ticker channel

The `ticker` channel provides real-time price updates every time a match happens. It batches updates in case of cascading matches, greatly reducing bandwidth requirements.

```
{
    "type": "ticker",
    "trade_id": 20153558,
    "sequence": 3262786978,
    "time": "2017-09-02T17:05:49.250000Z",
    "product_id": "BTC-USD",
    "price": "4388.01000000",
    "side": "buy", // Taker side
    "last_size": "0.03000000",
    "best_bid": "4388",
    "best_ask": "4388.01"
}
```

Please note that more information will be added to messages from this channel in the near future.

# The level2 channel

The easiest way to keep a snapshot of the order book is to use the `level2` channel. It guarantees delivery of all updates, which reduce a lot of the overhead required when consuming the `full` channel.

```
{
    "type": "snapshot",
    "product_id": "BTC-EUR",
    "bids": [["6500.11", "0.45054140"]],
    "asks": [["6500.15", "0.57753524"]]
}
```

When subscribing to the channel it will send a message with the type `snapshot` and the corresponding `product_id`. `bids` and `asks` are arrays of `[price, size]` tuples and represent the entire order book.

```
{
    "type": "l2update",
    "product_id": "BTC-EUR",
    "changes": [
        ["buy", "6500.09", "0.84702376"],
        ["sell", "6507.00", "1.88933140"],
        ["sell", "6505.54", "1.12386524"],
        ["sell", "6504.38", "0"]
    ]
}
```

Subsequent updates will have the type `l2update`. The `changes` property of `l2update`s is an array with `[side, price, size]` tuples. Please note that `size` is the updated size at that price level, not a delta. A size of `"0"` indicates the price level can be removed.

# The user channel

This channel is a version of the `full` channel that only contains messages that include the authenticated user. Consequently, you need to be authenticated to receive any messages.

# The matches channel

If you are only interested in match messages you can subscribe to the `matches` channel. This is useful when you're consuming the remaining feed using the level 2 channel.

Please note that messages can be dropped from this channel. By using the heartbeat channel you can track the last trade id and fetch trades that you missed from the REST API.

# The full channel

The `full` channel provides real-time updates on orders and trades. These updates can be applied on to a level 3 order book snapshot to maintain an accurate and up-to-date copy of the exchange order book.

> **Note:** If you are maintaining a level 2 order book, please consider switching to the level 2 channel.

An algorithm to maintain an up-to-date level 3 order book is described below. Please note that you will rarely need to implement this yourself.

1. Send a `subscribe` message for the product(s) of interest and the `full` channel.
2. Queue any messages received over the websocket stream.
3. Make a REST request for the order book snapshot from the REST feed.
4. Playback queued messages, discarding sequence numbers before or equal to the snapshot sequence number.
5. Apply playback messages to the snapshot as needed (see below).
6. After playback is complete, apply real-time stream messages as they arrive.

> All `open` and `match` messages will always result in a change to the order book. Not all `done` or `change` messages will result in changing the order book. These messages will be sent for received orders which are not yet on the order book. Do not alter the order book for such messages, otherwise your order book will be incorrect.

The following messages are sent over the websocket stream in JSON format when subscribing to the `full` channel:

**RECEIVED**

```
{
    "type": "received",
    "time": "2014-11-07T08:19:27.028459Z",
    "product_id": "BTC-USD",
    "sequence": 10,
    "order_id": "d50ec984-77a8-460a-b958-66f114b0de9b",
    "size": "1.34",
    "price": "502.1",
    "side": "buy",
    "order_type": "limit"
}
```

```
{
    "type": "received",
    "time": "2014-11-09T08:19:27.028459Z",
    "product_id": "BTC-USD",
    "sequence": 12,
    "order_id": "dddec984-77a8-460a-b958-66f114b0de9b",
    "funds": "3000.234",
    "side": "buy",
    "order_type": "market"
}
```

A valid order has been received and is now active. This message is emitted for every single valid order as soon as the matching engine receives it whether it fills immediately or not.

The `received` message does not indicate a resting order on the order book. It simply indicates a new incoming order which as been accepted by the matching engine for processing. Received orders may cause `match` message to follow if they are able to begin being filled (taker behavior). Self-trade prevention may also trigger `change` messages to follow if the order size needs to be adjusted. Orders which are not fully filled or canceled due to self-trade prevention result in an `open` message and become resting orders on the order book.

Market orders (indicated by the `order_type` field) may have an optional `funds` field which indicates how much quote currency will be used to buy or sell. For example, a `funds` field of `100.00` for the `BTC-USD` product would indicate a purchase of up to `100.00 USD` worth of bitcoin.

**OPEN**

```
{
    "type": "open",
    "time": "2014-11-07T08:19:27.028459Z",
    "product_id": "BTC-USD",
    "sequence": 10,
    "order_id": "d50ec984-77a8-460a-b958-66f114b0de9b",
    "price": "200.2",
    "remaining_size": "1.00",
    "side": "sell"
}
```

The order is now open on the order book. This message will only be sent for orders which are not fully filled immediately. `remaining_size` will indicate how much of the order is unfilled and going on the book.

There will be no `open` message for orders which will be filled immediately. There will be no `open` message for market orders since they are filled immediately.

**DONE**

```
{
    "type": "done",
    "time": "2014-11-07T08:19:27.028459Z",
    "product_id": "BTC-USD",
    "sequence": 10,
    "price": "200.2",
    "order_id": "d50ec984-77a8-460a-b958-66f114b0de9b",
    "reason": "filled", // or "canceled"
    "side": "sell",
    "remaining_size": "0"
}
```

The order is no longer on the order book. Sent for all orders for which there was a received message. This message can result from an order being canceled or filled. There will be no more messages for this `order_id` after a done message. `remaining_size` indicates how much of the order went unfilled; this will be `0` for `filled` orders.

`market` orders will not have a `remaining_size` or `price` field as they are never on the open order book at a given price.

A `done` message will be sent for received orders which are fully filled or canceled due to self-trade prevention. There will be no `open` message for such orders. `done` messages for orders which are not on the book should be ignored when maintaining a real-time order book.

**MATCH**

```
{
    "type": "match",
    "trade_id": 10,
    "sequence": 50,
    "maker_order_id": "ac928c66-ca53-498f-9c13-a110027a60e8",
    "taker_order_id": "132fb6ae-456b-4654-b4e0-d681ac05cea1",
    "time": "2014-11-07T08:19:27.028459Z",
    "product_id": "BTC-USD",
    "size": "5.23512",
    "price": "400.23",
    "side": "sell"
}
```

A trade occurred between two orders. The aggressor or `taker` order is the one executing immediately after being received and the `maker` order is a resting order on the book. The `side` field indicates the maker order side. If the side is `sell` this indicates the maker was a sell order and the `match` is considered an up-tick. A `buy` side match is a down-tick.

If authenticated, and you were the taker, the message would also have the following fields:

```
taker_user_id: "5844eceecf7e803e259d0365",
user_id: "5844eceecf7e803e259d0365",
taker_profile_id: "765d1549-9660-4be2-97d4-fa2d65fa3352",
profile_id: "765d1549-9660-4be2-97d4-fa2d65fa3352"
```

**CHANGE**

```
{
    "type": "change",
    "time": "2014-11-07T08:19:27.028459Z",
    "sequence": 80,
    "order_id": "ac928c66-ca53-498f-9c13-a110027a60e8",
    "product_id": "BTC-USD",
    "new_size": "5.23512",
    "old_size": "12.234412",
    "price": "400.23",
    "side": "sell"
}
```

```
{
    "type": "change",
    "time": "2014-11-07T08:19:27.028459Z",
    "sequence": 80,
    "order_id": "ac928c66-ca53-498f-9c13-a110027a60e8",
    "product_id": "BTC-USD",
    "new_funds": "5.23512",
    "old_funds": "12.234412",
    "price": "400.23",
    "side": "sell"
}
```

An order has changed. This is the result of self-trade prevention adjusting the order size or available funds. Orders can only decrease in size or funds. `change` messages are sent anytime an order changes in size; this includes resting orders (open) as well as received but not yet open. `change` messages are also sent when a new market order goes through self trade prevention and the `funds` for the market order have changed.

`change` messages for received but not yet open orders can be ignored when building a real-time order book. The `side` field of a change message and `price` can be used as indicators for whether the change message is relevant if building from a level 2 book.

Any `change` message where the price is `null` indicates that the `change` message is for a market order. Change messages for limit orders will always have a price specified.

#### ACTIVATE

An activate message is sent when a stop order is placed. When the stop is triggered the order will be placed and go through the order lifecycle.


```
{
  "type": "activate",
  "product_id": "test-product",
  "timestamp": "1483736448.299000",
  "user_id": "12",
  "profile_id": "30000727-d308-cf50-7b1c-c06deb1934fc",
  "order_id": "7b52009b-64fd-0a2a-49e6-d8a939753077",
  "stop_type": "entry",
  "side": "buy",
  "stop_price": "80",
  "size": "2",
  "funds": "50",
  "taker_fee_rate": "0.0025",
  "private": true
}
```


# FIX API

FIX (Financial Information eXchange) is a standard protocol which can be used to enter orders, submit cancel requests, and receive fills. Users of the FIX API will typically have existing software using FIX for order management. Users who are not familiar with FIX should first consider using the REST API.

#### FIX API ENDPOINT URL

`tcp+ssl://fix.pro.coinbase.com:4198`

Resend requests are not supported. Every connection establishes a new session and a new set of session sequence numbers.

# Connectivity

Before logging onto a FIX session, clients must establish a secure connection to the FIX gateway (`fix.pro.coinbase.com:4198`). If your FIX implementation does not support establishing a TCP SSL connection natively, you will need to setup a local proxy such as stunnel to establish a secure connection to the FIX gateway. See the SSL Tunnels section for more details and examples.

Coinbase Pro **does not** support static IP addresses. If your firewall rules require a static IP address, you will need to create a TCP proxy server with a static IP address which is capable of resolving an IP address using DNS.

If connecting from servers outside of AWS which require firewall rules, please use the AWS provided resources to determine how best to whitelist AWS IP ranges.

Routine disconnects occur every Monday at 1 PM Pacific Time. At that time, a **Logout** message will be sent from the server to indicate the session is ending.

# Messages

The baseline specification for this API is FIX 4.2. There are additional tags from later versions of FIX, and custom tags in the high number range as allowed by the standard.

A standard header must be present at the start of every message in both directions.

| Tag | Name | Description |
|---|---|---|
| 8 | BeginString | Must be `FIX.4.2` |
| 49 | SenderCompID | Client API key (on messages from the client) |
| 56 | TargetCompID | Must be `Coinbase` (on messages from the client) |

## Logon (A)

```
// create a new Logon message
var logon = new Msgs.Logon();
logon.SendingTime = new Date();
logon.HeartBtInt = 30;
logon.EncryptMethod = 0;
logon.passphrase = '...';

var presign = [
    logon.SendingTime,
    logon.MsgType,
    session.outgoing_seq_num,
    session.sender_comp_id,
    session.target_comp_id,
    passphrase
].join('\x01');

// add the presign string to the RawData field of the Logon message
logon.RawData = sign(presign, secret);

// send the logon message to the server
session.send(logon);

function sign(what, secret) {
    var key = Buffer(secret, 'base64');
    var hmac = crypto.createHmac('sha256', key);
    return hmac.update(what).digest('base64');
}
```

Sent by the client to initiate a session, and by the server as an acknowledgement. Only one session may exist per connection; sending a Logon message within an established session is an error.

| Tag | Name | Description |
|---|---|---|
| 98 | EncryptMethod | Must be `0` (None) |
| 108 | HeartBtInt | Must be `30` (seconds) |
| 554 | Password | Client API passphrase |
| 96 | RawData | Client message signature (see below) |
| 8013 | CancelOrdersOnDisconnect | `Y`: Cancel all open orders for the current profile; `S`: Cancel open orders placed during session |
| 9406 | DropCopyFlag | If set to `Y`, execution reports will be generated for all user orders (defaults to `Y`) |

The Logon message sent by the client must be signed for security. The signing method is described in Signing a Message. The prehash string is the following fields joined by the FIX field separator (ASCII code 1):

`SendingTime, MsgType, MsgSeqNum, SenderCompID, TargetCompID, Password`.

There is no trailing separator. The RawData field should be a `base64` encoding of the HMAC signature.

A single API key must not be used in multiple connections at the same time. To establish multiple FIX connections, please generate a new API key for each one. A maximum of 10 connections can be established.

---

# Logout (5)

Sent by either side to initiate session termination. The side which receives this message first should reply with the same message type to confirm session termination. Closing a connection without logging out of the session first is an error.

---

# New Order Single (D)

Sent by the client to enter an order.

| Tag | Name | Description |
|---|---|---|
| 21 | HandlInst | Must be `1` (Automated) |
| 11 | ClOrdID | UUID selected by client to identify the order |
| 55 | Symbol | E.g. `BTC-USD` |
| 54 | Side | Must be `1` to buy or `2` to sell |
| 44 | Price | Limit price (e.g. in USD) (Limit order only) |
| 38 | OrderQty | Order size in base units (e.g. BTC) |
| 152 | CashOrderQty | Order size in quote units (e.g. USD) (Market order only) |
| 40 | OrdType | Must be `1` for Market, `2` for Limit, `3` for Stop Market, or `4` for Stop Limit |
| 99 | StopPx | Stop price for order |

| Tag | Name | Description |
|---|---|---|
| 59 | TimeInForce | Must be a valid TimeInForce value. See the table below (Limit order only) |
| 7928 | SelfTradePrevention | Optional, see the table below |

#### SELFTRADEPREVENTION VALUES

| Value | Description |
|---|---|
| D | Decrement and cancel (the default) |
| O | Cancel resting order |
| N | Cancel incoming order |
| B | Cancel both orders |

If an order is decremented due to self-trade prevention, an Execution Report will be sent to the client with ExecType=D indicating unsolicited OrderQty reduction (i.e. partial cancel).

See the self-trade prevention documentation for more details about this field.

#### TIMEINFORCE VALUES

| Value | Description |
|---|---|
| 1 | Good Till Cancel |
| 3 | Immediate or Cancel |
| 4 | Fill or Kill |
| P | Post-Only |

The post-only flag (`P`) indicates that the order should only make liquidity. If any part of the order results in taking liquidity, the order will be rejected and no part of it will execute. Open Post-Only orders will be treated as Good Till Cancel.

See the time in force documentation for more details about these values.

#### ERRORS

If a trading error occurs (e.g. user has insufficient funds), an ExecutionReport with `ExecType=8` is sent back, signifying that the order was rejected.

---

# Order Cancel Request (F)

Sent by the client to cancel an order.

| Tag | Name | Description |
|---|---|---|
| 11 | ClOrdID | UUID selected by client for the order |
| 37 | OrderID | OrderID from the ExecutionReport with OrdStatus=New (39=0) |
| 41 | OrigClOrdID | ClOrdID of the order to cancel (originally assigned by the client) |
| 55 | Symbol | Symbol of the order to cancel (must match Symbol of the Order) |

#### CLORDID

Use of the ClOrdID is not available after reconnecting or starting a new session. You should use the OrderID obtained via the ExecutionReport once available.

---

# Order Status Request (H)

Sent by the client to obtain information about pending orders.

| Tag | Name | Description |
|---|---|---|
| 37 | OrderID | OrderID of order(s) to be sent back. Can be equal to `*` (wildcard) to send back all pending orders |

#### RESPONSE

The response to an Order Status Request is a series of ExecutionReports with `ExecType=I`, each representing one open order belonging to the user. If the user has no open orders, a single ExecutionReport is sent back with `OrderID=0`.

---

# Execution Report (8)

Sent by the server when an order is accepted, rejected, filled, or canceled. Also sent when the user sends an `OrderStatusRequest`.

| Tag | Name | Description |
|---|---|---|
| 11 | ClOrdID | Only present on order acknowledgements, ExecType=New (150=0) |
| 37 | OrderID | OrderID from the ExecutionReport with ExecType=New (150=0) |
| 55 | Symbol | Symbol of the original order |
| 54 | Side | Must be `1` to buy or `2` to sell |
| 32 | LastShares | Amount filled (if ExecType=1). Also called LastQty as of FIX 4.3 |
| 44 | Price | Price of the fill if ExecType indicates a fill, otherwise the order price |
| 38 | OrderQty | OrderQty as accepted (may be less than requested upon self-trade prevention) |
| 152 | CashOrderQty | Order size in quote units (e.g. USD) (Market order only) |
| 60 | TransactTime | Time the event occurred |
| 150 | ExecType | May be `1` (Partial fill) for fills, `D` for self-trade prevention, etc. |
| 39 | OrdStatus | Order status as of the current message |
| 103 | OrdRejReason | Insufficient funds=`3`, Post-only=`8`, Unknown error=`0` |
| 136 | NoMiscFees | `1` (Order Status Request response only) |
| 137 | MiscFeeAmt | Fee (Order Status Request response only) |
| 139 | MiscFeeType | `4` (Exchange fees) (Order Status Request response only) |
| 1003 | TradeID | Product unique trade id |
| 1057 | AggressorIndicator | `Y` for taker orders, `N` for maker orders |

#### EXECTYPE VALUES

| ExecType | Description |
|---|---|
| 0 | New Order |
| 1 | Fill |
| 3 | Done |
| 4 | Canceled |
| 7 | Stopped |
| 8 | Rejected |
| D | Order Changed |
| I | Order Status |

# Order Cancel Reject (9)

Sent by the server when an Order Cancel Request cannot be satisfied, e.g. because the order is already canceled or completely filled.

| Tag | Name | Description |
|---|---|---|
| 11 | ClOrdID | As on the cancel request |
| 37 | OrderID | As on the cancel request |
| 41 | OrigClOrdID | As on the cancel request |
| 39 | OrdStatus | 4 if too late to cancel |
| 102 | CxlRejReason | 1 if the order is unknown |
| 434 | CxlRejResponseTo | 1 (Order Cancel Request) |

# Reject (3)

Sent by either side upon receipt of a message which cannot be processed, e.g. due to missing fields or an unsupported message type.

| Tag | Name | Description |
|---|---|---|
| 45 | RefSeqNum | MsgSeqNum of the rejected incoming message |
| 371 | RefTagID | Tag number of the field which caused the reject (optional) |
| 372 | RefMsgType | MsgType of the rejected incoming message |
| 58 | Text | Human-readable description of the error (optional) |
| 373 | SessionRejectReason | Code to identify reason for reject |

#### SESSIONREJECTREASON VALUES

The following values can be sent by the server.

| Value | Description |
|---|---|
| 1 | Required tag missing |
| 5 | Value is incorrect (out of range) for this tag |
| 6 | Incorrect data format for value |
| 11 | Invalid MsgType (35) |

## Heartbeat (0)

Sent by both sides if no messages have been sent for HeartBtInt seconds as agreed during logon. May also be sent in response to a Test Request.

| Tag | Name | Description |
|---|---|---|
| 112 | TestReqID | Copied from the Test Request, if any |

## Test Request (1)

May be sent at any time by either side.

| Tag | Name | Description |
|---|---|---|
| 112 | TestReqID | Free text |

# SSL Tunnels

`fix.pro.coinbase.com:4198` only accepts TCP connections secured by SSL. If your FIX client library cannot establish an SSL connection natively, you will need to run a local proxy that will establish a secure connection and allow unencrypted local connections.

## Stunnel Configuration

This is an example configuration file for stunnel to listen on a port locally and proxy unencrypted TCP connections to the encrypted SSL connection. The service name (`Coinbase`) and the accept port (`4197`) may be changed to any suitable values.

```
[Coinbase]
client = yes
accept = 4197
connect = fix.pro.coinbase.com:4198
verify = 4
CAfile = /example/path/to/fix.pro.coinbase.com.pem
```

When stunnel is started with the above configuration file, it will run in the background. On Unix-like systems the option `foreground = yes` may be specified at the top of the file to avoid running in the background. For testing it may be easier to use foreground mode, or

to specify the top-level `output` option as a file path where stunnel will write log messages.

> The stunnel configuration must include either `verify=3` or `verify=4` to enable client certificate pinning. The exchange certificate is available via `pro.coinbase.com` and must be installed in a secure (not openly writable) directory on the client system which is specified in the stunnel configuration file as `CAfile`.

If your system has OpenSSL installed, you can run this command to download the certificate:

```
openssl s_client -showcerts -connect fix.pro.coinbase.com:4198 < /dev/null | openssl x509 -outform PEM > fix.pro.coinbase.com.pem
```

> To connect to sandbox, change the url to `fix-public.sandbox.pro.coinbase.com` to download the appropriate certificate.