Welcome

This page will help you get started with Alpaca Docs. You'll be up and running in a jiffy!

Suggest Edits

Welcome to Alpaca 

Alpaca offers simple, modern API-first solutions to enable anyone, either individuals or businesses, to connect applications and build algorithms to buy and sell stocks or crypto.

Whether you are launching an app to access the US equities market or deploy algorithmic trading-strategies with stocks and crypto, Alpaca has an API for you.

- Build trading apps and brokerage services for your end users. Tailored for businesses such as trading apps, challenger banks, etc. - **Start here**
- Stock trading for individuals and business accounts. Built for retail, algorithmic and proprietary traders. - **Start here**
- Access real-time market pricing data and up to 6+ years worth of historical data for stocks and crypto. - **Start here**
- Develop applications on Alpaca's platform using OAuth2. Let any user with an Alpaca brokerage account connect to your app. - **Start here**

If you are not a developer and an API is not for you, we also enable users to trade on our responsive web dashboard.

Stay tuned for our API updates as we have on roadmap plans for options, futures, FX, and much more!

About Alpaca

Suggest Edits

About Alpaca

History & Founders

Alpaca is a technology company headquartered in Silicon Valley that builds a simple and modern API for stock and crypto trading. Our Brokerage services are provided by Alpaca Securities LLC, a member of FINRA/SIPC. We are a team of diverse background individuals with deep financial and technology expertise,

backed by some of the top investors in the industry globally. We are proud to be supported by the love of enthusiastic community members on various platforms.

Alpaca's globally distributed team consists of developers, traders, and brokerage business specialists, who collectively have decades of financial services and technology industry experience at organizations such as FINRA, Apple, Wealthfront, Robinhood, EMC, Cloudera, JP Morgan, and Lehman Brothers. Alpaca is co-founded and led by Yoshi Yokokawa (CEO) and Hitoshi Harada (CPO).

Our investors include a group of well-capitalized investors including Portage Ventures, Spark Capital, Tribe Capital, Social Leverage, Horizons Ventures, Elderidge, and Y Combinator as well as highly experienced industry angel investors such as Joshua S. Levine (former CTO/COO of ETRADE), Nate Rodland (former COO of Robinhood & GP of Elefund), Patrick O'Shaughnessy ("Invest Like the Best" podcast host & Partner of Positive Sum), Jacqueline Reses (former Executive Chairman of Square Financial Services), Asiff Hirjii (former President/COO of Coinbase), Aaron Levie (CEO of Box), and founders of leading Fintech companies including Plaid and Wealthsimple.

Vision

Allow 8+ billion people on the planet to access financial markets.

We are committed to providing a secure, reliable and compliant platform for anyone who wants to build their own trading strategies, asset management automation, trading and robo-advisor apps, new brokerage services, investment advisory services and investment products. We value having a variety of developers who create exciting and innovative projects with our API.

What Services Does Alpaca Provide?

Alpaca provides trading and clearing services for US equities under its subsidiary Alpaca Securities LLC. We currently support stocks, ETFs listed in the US public exchanges (NMS stocks) and cryptocurrencies. Support for other asset classes, such as options, futures, FX, private equities, and international equities are on our roadmap.

We work with retail traders and institutional investors directly, and also work with app developers, broker-dealers globally, investment advisors and fintech

companies that offer US stock investing features to their end customers, both in and out of the United States.

To serve our customers, we provide Web API, Web dashboards, market data, paper trading simulation, API sandbox environment, and community platforms.

Alpaca API Platform

Suggest Edits

Why API?

Alpaca's features to access financial markets are provided primarily via API. We believe API is the means to interact with services such as ours and innovate your business. Our API is designed to fit your needs and we continue to build what you need.

REST, SSE and Websockets

Our API is primarily built in the REST style. It is a simple and powerful way to integrate with our services.

In addition to the REST API which replies via synchronous communication, our API includes an asynchronous event API which is based on WebSocket and SSE, or Server-Sent Events. As many types of events occur in the financial markets (orders fill based on the market movement, cash settles after some time, etc), this event-based API helps you get updates instantly and provide the best user experiences to your customers.

Architecture

Alpaca's platform consists of APIs, Web dashboards, trade simulator, sandbox environment, authentication services, order management system, trading routing, back office accounting and clearing system, and all of these components are built in-house from the ground up with modern architecture.

The Alpaca platform is currently hosted on the Google Cloud Platform in the us-east4 region. The site is connected with dedicated fiber lines to a data center in Secaucus, NJ, to cross-connect with various market venues.

Under the hood, Alpaca works with various third parties. We work with Velox Clearing and Vision Financial for equities trade clearing and settlement on DTCC. Cash transfers and custody are primarily provided by BMO Harris and Silicon Valley Bank. Citadel Securities, Virtu America, Jane Street, UBS, and other execution providers provide execution services for our customer orders. We integrate with ICE Data Services for various kinds of market data.

API Updates & Upgrades

In an effort to continuously provide value to our developers, Alpaca frequently performs updates and upgrades to our API.

We've added the following sections to our docs in order to help make sure that as a developer you know what to expect, when to expect, and how to properly handle certain scenarios .

Backwards Compatible Changes

You should expect any of the following kind of changes that we make to our API to be considered a backwards compatible change:

- Adding new or similarly named APIs
- Adding new fields to already defined models and objects such as API return objects, nested objects, etc. (Example: adding a new code field to error payloads)
- Adding new items to defined sets or enumerations such as statuses, supported assets, etc. (Example: adding new account status to a set of all)
- Enhancing ordering on how certain lists get returned
- Supporting new HTTP versions (HTTP2, QUIC)
- Adding new HTTP method(s) for an existing endpoint
- Expecting new HTTP request headers (eg. new authentication)
- Sending new HTTP headers (eg. HTTP caching headers, gzip encoding, etc.)
- Increasing HTTP limits (eg. Nginx large-client-header-buffers)
- Increasing rate limits
- Supporting additional SSL/TLS versions

Generally, as a rule of thumb, any append or addition operation is considered a backwards compatible update and does not need an upfront communication. These updates should be backwards compatible with existing interfaces and not cause any disruption to any clients calling our APIs.

Breaking Changes

When and if Alpaca decides to perform breaking changes to our APIs the following should be expected:

- Upfront communication with sufficient time to allow developers to be able to react to new upcoming changes
- Our APIs are versioned, if breaking changes are intended we will generally bump the API version. For example, a route might go from being `/v1/accounts/{account_id}` to `/v2/accounts/{account_id}` if we had to make a breaking change to either the parameters it can take or its return structure

SDKs and Tools

Suggest Edits

Official Client SDKs

Alpaca provides and supports the following open-source SDKs in a number of languages. You can leverage these libraries to easily access our API in your own application code or your trading scripts.

- Python: alpaca-py / PyPI
- .NET/C#: alpaca-trade-api-csharp / NuGet
- Node: alpaca-trade-api-js / npm
- Go: alpaca-trade-api-go
- Python (legacy): alpaca-trade-api-python / PyPI

Community-Made SDKs

In addition to the SDKs directly supported by Alpaca, individual members of our community have created and contributed their own wrappers for these other languages. We are providing these links as a courtesy to the community and to our users who are looking for the API wrapper in other languages or variants.

Please be sure to carefully review any code you use to access our financial trading API and/or trust your account credentials to.

Made your own wrapper for a language not listed? Join our community Slack and let us know about it!

- C++: alpaca-trade-api-cpp
- Java: alpaca-java
- Node.js (TypeScript): alpaca-ts
- R: alpaca-for-r
- Rust: apca (SDK) & apcacli (CLI)
- Scala: Alpaca Scala
- Ruby: alpaca-trade-api
- Elixir: alpaca_elixir

Additional Resources

- Suggest Edits

Alpaca Learn

- We post regular content on our Alpaca Learn resource site where you can find the latest market updates, development tools and tips and much more to help you along your journey developing with Alpaca. For more click here.

Blog

- Dont miss a beat and find the latest updates from Alpaca in our blog. For more click here.

Slack Community

- We have an active community on Slack with developers and traders from all over the world. For Broker API we have a dedicated channel #broker-api for you to discuss with the rest of the community building new products using Broker API. You will be automatically added to `#announcements`, `#community`, `#feedback`, and `#q-and-a`. To join click here.

Forum

- We have set up a community forum to discuss topics ranging from integration, programming, API questions, market data, etc. The forum is also a place to find up-to-date announcements about Alpaca and its features. The forum is the recommended place for discussion, since it has

more advanced indexing and search functionality compared to our other community channels. For more click here.

- **Support**
- Have questions? Need help? Check out our support page for FAQs and to get in contact with our team. For more click here.
- **Systems Status**
- Stay up to date with Alpaca services status and any updates on outages. For more click here.
- **Disclosures**
- To view our disclosures library click here.

About Broker API

This is the documentation about Broker API that helps you build trading apps and brokerage services for your end users. If you are looking to build your own trading bots and algos, read the Trading API documentation. With Alpaca Broker API, you can build the full brokerage experiences for your end users around account opening, funding and trading. This document describes all you need to know to build your trading app.

Suggest Edits

Broker API Use Cases

There are several different use cases for Broker API integration. Below are some common ones, but please do not hesitate to reach out to our sales team if you have a different case in mind. We want our platform to encourage a broad range of use cases.

- Trading/investing app (non-financial institution)
- Broker dealer (fully-disclosed, omnibus)
- Registered Investment Advisor (RIA)

We support most use cases internationally.

Depending on the case, the API methods you want to use could vary. For example, the omnibus broker-dealer case never uses API to open a customer account since the trading accounts are created upfront and you will submit orders to them, and manage your end customer accounting on your end. More details on each use case are described in the following sections.

Getting Started with Broker API

This guide is going to help you set everything up in a sandbox environment to get you up and running with Broker API.

Suggest Edits

The sandbox environment acts as a parallel environment where you can test our APIs safely without sending any real trades to the market. All prices, and execution times (i.e. market hours) hold true in sandbox and production.

You can either follow the steps below to test specific calls within the broker dashboard or access the Postman collection to view and test all possible requests in one place.

Postman Collection

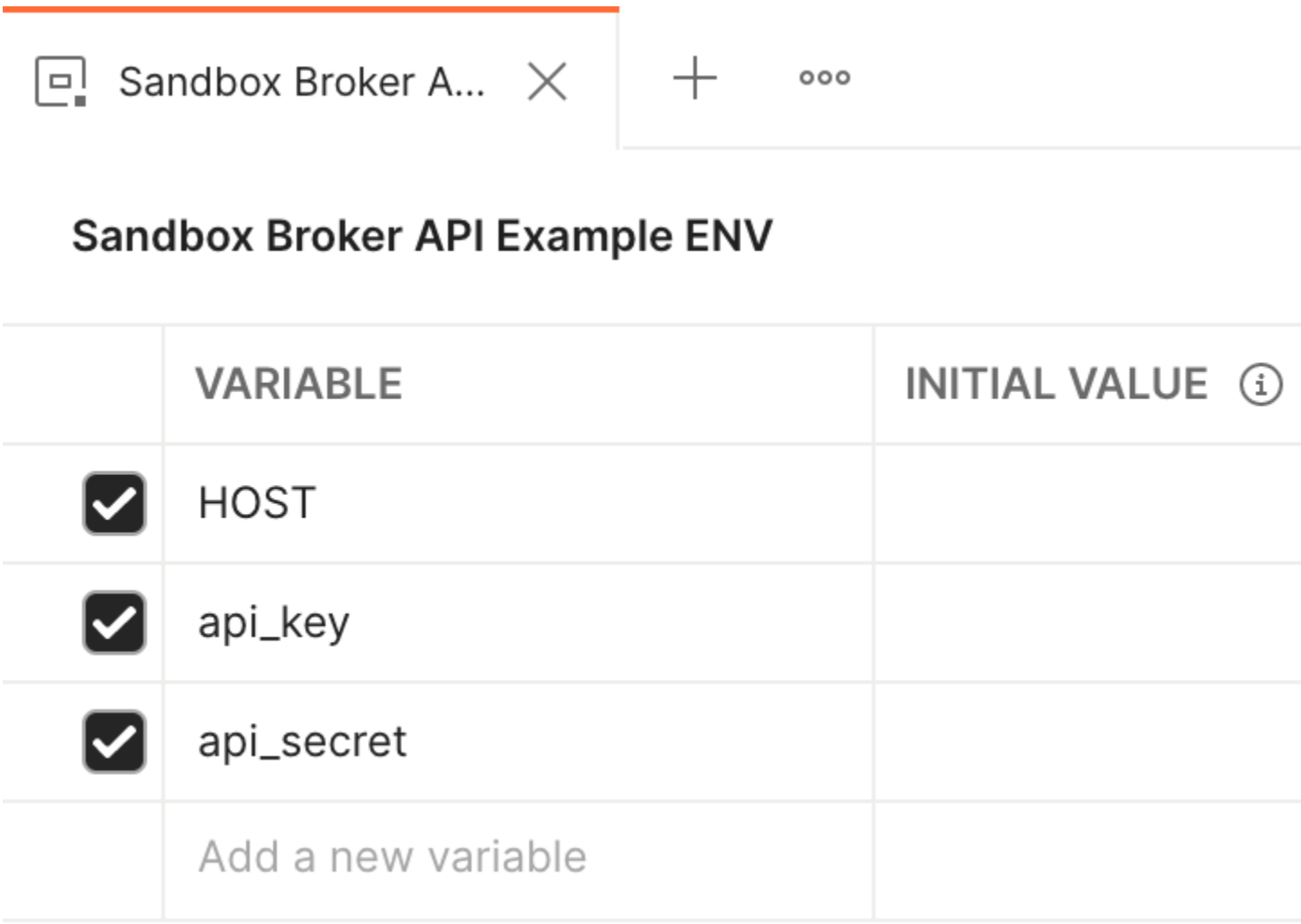
To get started with the Broker API Postman Collection you can either access the Alpaca Workspace on Postman to fork the collection or import the file below directly to your own workspace.

Fork Broker API Collection on Postman

Refer to this tutorial to learn how to fork the collection and sample environment and get started with making calls right away. We recommend following this method so your collection stays up to date with the changes we make to the API.

Import Broker API Collection

1. Download the Broker API API Collection
2. Import the file into Postman (File -> Import..)
3. Create a Postman environment with the following variables. Be sure to select the environment in the upper right hand corner like pictured below.



4. Send one of the defined HTTP requests while the created environment is selected.

Testing on Broker Dashboard (Brokerdash)

0. Setting up Broker API on Brokerdash

API Keys

When you sign up for an account at Alpaca you will receive an `API_KEY` and `API_SECRET`, please make sure you store those somewhere safe. Broker API must authenticate using HTTP Basic authentication. Use your correspondent `API_KEY` and `API_SECRET` as the username and password. The format

is key:secret. Encode the string with base-64 encoding, and you can pass it as an authentication header.

Live Environment

We have provided in our dashboard an API tool that uses your API key credentials to send requests and receive responses straight from your browser.

Simply navigate to API/Devs > Live Testing and try out our APIs.

Making Your First Request

At this point we can assume that you haven't created any accounts yet, but one of the first API calls you can make is `GET /v1/assets`, which doesn't require a request body and will give you all the assets available at Alpaca.
The response would contain an array of assets, with the first one being Agilent Technologies Inc. as of 2021-05-17

JSON

```
{
        "id": "7595a8d2-68a6-46d7-910c-6b1958491f5c",
        "class": "us_equity",
        "exchange": "NYSE",
        "symbol": "A",
        "name": "Agilent Technologies Inc.",
        "status": "active",
        "tradable": true,
        "marginable": true,
        "shortable": true,
        "easy_to_borrow": true,
        "fractionable": true
},
```


1. Create an Account

One of the first things you would need to do using Broker API is to create an account for your end user. Depending on the type of setup you have with Alpaca (Fully-Disclosed, Omnibus or RIA) the requirements might differ.

Below is a sample request to create an account for a Fully-Disclosed setup:

JSON

```
{
  "contact": {
    "email_address": "awesome_alpaca@example.com",
    "phone_number": "555-666-7788",
    "street_address": ["20 N San Mateo Dr"],
    "city": "San Mateo",
```

```
    "state": "CA",
    "postal_code": "94401",
    "country": "USA"
  },
  "identity": {
    "given_name": "John",
    "family_name": "Doe",
    "date_of_birth": "1990-01-01",
    "country_of_citizenship": "USA",
    "country_of_birth": "USA",
    "country_of_tax_residence": "USA",
    "funding_source": ["employment_income"]
  },
  "disclosures": {
    "is_control_person": false,
    "is_affiliated_exchange_or_finra": false,
    "is_politically_exposed": false,
    "immediate_family_exposed": false
  },
  "agreements": [
    {
      "agreement": "margin_agreement",
      "signed_at": "2020-09-11T18:09:33Z",
      "ip_address": "185.13.21.99"
    },
    {
      "agreement": "account_agreement",
      "signed_at": "2020-09-11T18:13:44Z",
      "ip_address": "185.13.21.99"
    },
    {
      "agreement": "customer_agreement",
      "signed_at": "2020-09-11T18:13:44Z",
      "ip_address": "185.13.21.99"
    }
  ],
  "documents": [
    {
      "document_type": "cip_result",
      "content": "VGhlcmUgYXJlIG5vIHdpbGQgYWxwYWNhcy4=",
      "mime_type": "application/pdf"
    },
    {
      "document_type": "identity_verification",
      "document_sub_type": "passport",
      "content": "QWxwYWNhcyBjYW5ub3QgbGl2ZSBhbG9uZS4=",
      "mime_type": "image/jpeg"
    }
  ],
  "trusted_contact": {
    "given_name": "Jane",
    "family_name": "Doe",
    "email_address": "jane.doe@example.com"
  }
}
```


If successful, the reponse would be

JSON

```
{
  "id": "b9b19618-22dd-4e80-8432-fc9e1ba0b27d",
  "account_number": "935142145",
  "status": "APPROVED",
  "currency": "USD",
  "last_equity": "0",
  "created_at": "2021-05-17T09:53:17.588248Z"
}
```


2. Fund an Account via ACH

Creating an ACH Relationship

In order to virtually fund an account via ACH we must first establish the ACH Relationship with the account.

We will be using the following endpoint `POST /v1/accounts/{account_id}/ach_relationships` replacing the `account_id` with `b9b19618-22dd-4e80-8432-fc9e1ba0b27d`

JSON


```
{
  "account_owner_name": "Awesome Alpaca",
  "bank_account_type": "CHECKING",
  "bank_account_number": "32131231abc",
  "bank_routing_number": "121000358",
  "nickname": "Bank of America Checking"
}
```


Please make sure that the formatting for `bank_account_number` and `bank_routing_number` are in the correct format.
If successful you will receive an `ach_relationship` object like this:

JSON


```
{
  "id": "c9b420e0-ae4e-4f39-bcbf-649b407c2129",
  "account_id": "b9b19618-22dd-4e80-8432-fc9e1ba0b27d",
  "created_at": "2021-05-17T09:54:58.114433723Z",
  "updated_at": "2021-05-17T09:54:58.114433723Z",
  "status": "QUEUED",
  "account_owner_name": "Awesome Alpaca",
  "bank_account_type": "CHECKING",
  "bank_account_number": "32131231abc",
  "bank_routing_number": "121000358",
  "nickname": "Bank of America Checking"
}
```


Initially you will receive a `status = QUEUED`.
However, if you make a `GET/v1/accounts/{account_id}/ach_relationships`, after ~1 minute you should see `status = APPROVED`.

Making a Virtual ACH Transfer

Now that you have an existing ACH relationship between the account and their bank, you can fund the account via ACH using the following endpoint `POST/v1/accounts/{account_id}/transfers` using the `relationship_id` we got in the response of the previous section.

JSON

```
{
  "transfer_type": "ach",
  "relationship_id": "c9b420e0-ae4e-4f39-bcbf-649b407c2129",
  "amount": "1234.567",
  "direction": "INCOMING"
}
```

The response you should get would look like this.

JSON

```
{
  "id": "750d8323-19f6-47d5-8e9a-a34ed4a6f2d2",
  "relationship_id": "c9b420e0-ae4e-4f39-bcbf-649b407c2129",
  "account_id": "b9b19618-22dd-4e80-8432-fc9e1ba0b27d",
  "type": "ach",
  "status": "QUEUED",
  "amount": "1234.567",
  "direction": "INCOMING",
  "created_at": "2021-05-17T09:56:05.445592162Z",
  "updated_at": "2021-05-17T09:56:05.445592162Z",
  "expires_at": "2021-05-24T09:56:05.445531104Z"
}
```

After around 10-30 minutes (to simulate ACH delay) the transfer should reflect on the user's balance via a cash deposit activity (CSD) viewed via this endpoint `GET v1/accounts/activities/CSD\?account_id\={account_id}`

3. Journaling Between Accounts

In addition to transfer and funding via ACH and wire, we have enabled organizations to directly fund their Firm Accounts and then journal from those to user's accounts in order to simulate near instantaneous funding.

Introducing the Firm Account

Each team will come with a firm account in sandbox that is pre-funded for $50,000. You can use this account to simulate funding to your users or use it for rewards programs to fuel your app's growth.

To illustrate our example, the Sweep account for this sandbox account looks like this

JSON

```
{
```

```
  "id": "8f8c8cee-2591-4f83-be12-82c659b5e748",
  "account_number": "927721227",
  "status": "ACTIVE",
  "currency": "USD",
  "last_equity": "45064.36",
  "created_at": "2021-03-03T17:50:06.568149Z"
}
```


Journaling Cash

In the case of a signup reward, or simply attempting to simulate instant funding, journaling funds between your firm balance with Alpaca and the end user's brokerage account is the best way.

You can simply pass in a request with `entry_type =JNLC` and choose the amount you want to journal to the user.

4. Passing an Order

The most common use case of Alpaca Broker API is to allow your end users to trade on the stock market. To do so simply pass in to `POST /v1/trading/accounts/{account_id}/orders` and again replacing the `account_id` with `b9b19618-22dd-4e80-8432-fc9e1ba0b27d`

JSON


```
{
  "symbol": "AAPL",
  "qty": 0.42,
  "side": "buy",
  "type": "market",
  "time_in_force": "day"
}
```


Whatever the response from Alpaca would be (denoted by the status) you should receive an Order model in the response looking like this

JSON


```
{
  "id": "4c6cbac4-e17a-4373-b012-d446b20f9982",
  "client_order_id": "5a5e2660-88a7-410c-92c9-ab0c942df70b",
  "created_at": "2021-05-17T11:27:18.499336Z",
  "updated_at": "2021-05-17T11:27:18.499336Z",
  "submitted_at": "2021-05-17T11:27:18.488546Z",
  "filled_at": null,
  "expired_at": null,
  "canceled_at": null,
  "failed_at": null,
  "replaced_at": null,
  "replaced_by": null,
  "replaces": null,
  "asset_id": "b0b6dd9d-8b9b-48a9-ba46-b9d54906e415",
  "symbol": "AAPL",
```

```
  "asset_class": "us_equity",
  "notional": null,
  "qty": "0.42",
  "filled_qty": "0",
  "filled_avg_price": null,
  "order_class": "",
  "order_type": "market",
  "type": "market",
  "side": "buy",
  "time_in_force": "day",
  "limit_price": null,
  "stop_price": null,
  "status": "accepted",
  "extended_hours": false,
  "legs": null,
  "trail_percent": null,
  "trail_price": null,
  "hwm": null,
  "commission": "0"
}
```


5. Events (SSE)

You can always listen to any event changes to accounts, journals or orders via our Events SSE.

An example for a journal update via this endpoint GET/v1/events/journal/updates where it shows all the different stages the journal id = 2f144d2a-91e6-46ff-8e37-959a701cc58d is going through.


```
data: {"at":"2021-05-
07T10:28:23.163857Z","entry_type":"JNLC","event_id":1406,"journal_id":"2f144d2a-
91e6-46ff-8e37-959a701cc58d","status_from":"","status_to":"queued"}

data: {"at":"2021-05-
07T10:28:23.468461Z","entry_type":"JNLC","event_id":1407,"journal_id":"2f144d2a-
91e6-46ff-8e37-959a701cc58d","status_from":"queued","status_to":"pending"}

data: {"at":"2021-05-
07T10:28:23.522047Z","entry_type":"JNLC","event_id":1408,"journal_id":"2f144d2a-
91e6-46ff-8e37-959a701cc58d","status_from":"pending","status_to":"executed"}
```


You are now ready to explore more of Broker API!

Have a look at our API References and feel free to contact us anytime through Intercom on your Broker Dashboard!

Request ID

All Broker API endpoint provides a unique identifier of the API call in the response header with `X-Request-ID` key, the Request ID helps us to identify the call chain in our system.

Make sure you provide the Request ID in all support requests that you created, it could help us to solve the issue as soon as possible. Request ID can't be queried in other endpoints, that is why we suggest to persist the recent Request IDs.

Shell

```
$ curl -v https://broker-api.sandbox.alpaca.markets/v1/accounts
...
> GET /v1/accounts HTTP/1.1
> Host: broker-api.sandbox.alpaca.markets
> User-Agent: curl/7.88.1
> Accept: */*
>
< HTTP/1.1 403 Forbidden
< Date: Fri, 25 Aug 2023 09:10:03 GMT
< Content-Type: application/json
< Content-Length: 26
< Connection: keep-alive
< X-Request-ID: 65ddd35ed1b3433dbf29d11f6d932c88
<
...
```

Use Cases

Suggest Edits

There are several different use cases for Broker API integration. Below are some common ones, but please do not hesitate to reach out to our sales team if you have a different case in mind. We want our platform to encourage a broad range of use cases.

- Trading/investing app (non-financial institution)
- Broker dealer (fully-disclosed, omnibus)
- Registered Investment Advisor (RIA)

We support most use cases internationally.

Depending on the case, the API methods you want to use could vary. For example, the omnibus broker-dealer case never uses API to open a customer account since the trading accounts are created upfront and you will submit orders to them, and manage your end customer accounting on your end. More details on each use case are described in the following sections.

Trading/Investing App

(Non-financial Institution)

As a technology company, you are building a brand-new trading or investing app. You could also be an established fintech firm who offers cash management services, a neobank, or maybe a payroll company without a financial institution status/license. Alpaca Broker API could be a solution for you. In this case, the account approval process is owned by Alpaca, and you own the user experience and most of the communications. Unless you have a robust CIP/KYC/AML program in place that has been vetted by Alpaca, your customers will go through Alpaca's CIP/KYC/AML process.

Alpaca will work hand-in-hand with you to market your service using Broker API in a compliant manner.

This partnership structure is only available in select countries. Please contact sales@alpaca.markets for further information.

Broker Dealer

Fully-Disclosed

You are a registered broker-dealer in your jurisdiction and you introduce your customers to Alpaca to establish individual accounts on a fully-disclosed basis. Alpaca receives each customer's information in order to open an account and provide our services.

Depending on your registration status and regulations in your jurisdiction, you may own the full end-to-end user experience from account opening, to trading, and reporting. You are responsible for maintaining a robust Customer Identification Program (CIP) and Know Your Customer (KYC) procedures, in accordance with Anti-Money Laundering (AML) regulations. To ensure compliance with applicable laws, Alpaca will conduct due diligence on your firm, including a thorough review of your CIP/KYC/AML program.

In this setup, you will use most of the API methods such as the Account, Transfer and Trading API. In addition, you can move cash and securities between your firm account and end-customer accounts using the Journal API to implement features such as reward programs.

Omnibus

You are a registered broker-dealer and you manage customer accounting, and two main trading accounts for the entire trading flow of your customers (one for long positions and one for short positions). In the omnibus setup, your end customer information (e.g. name, address) is not disclosed to Alpaca.

When submitting customer orders for your two trading accounts, you will indicate if the order is for the long or short position of each customer. To meet our regulatory requirements, you will be required to submit a "sub-tag" in each order to identify different customer's order flow. This can be just an arbitrary text string such as UUID or a customer ID number. Alpaca, as well as our trading execution venues, need to be able to review and account for all trading activity. Failure to submit the required trading order flow information may result in the suspension of the entire order flow.

As each end customer is not disclosed to Alpaca, you are responsible for all tax reporting in your local jurisdiction. If you are a non-US broker-dealer, this will likely require registration with the IRS as a Foreign Financial Institution (FFI) and get certified as a Qualified Intermediary (QI) to be certified to manage taxes on the US-sourced income by foreigners.

Registered Investment Advisor (RIA)

You are a SEC-registered RIA with customers either in or outside the US.

Similar to the Trading/Investing App approach described above, the end customer is introduced to Alpaca on an individual basis. The account approval process is owned by Alpaca, and you own the user experience and most of the communications. Unless you have a robust CIP/KYC/AML program in place that has been vetted by Alpaca, your customers will go through Alpaca's CIP/KYC/AML process.

The Account API works slightly differently from the fully-disclosed case for this setup. Please see the rest of API documentation for more details.

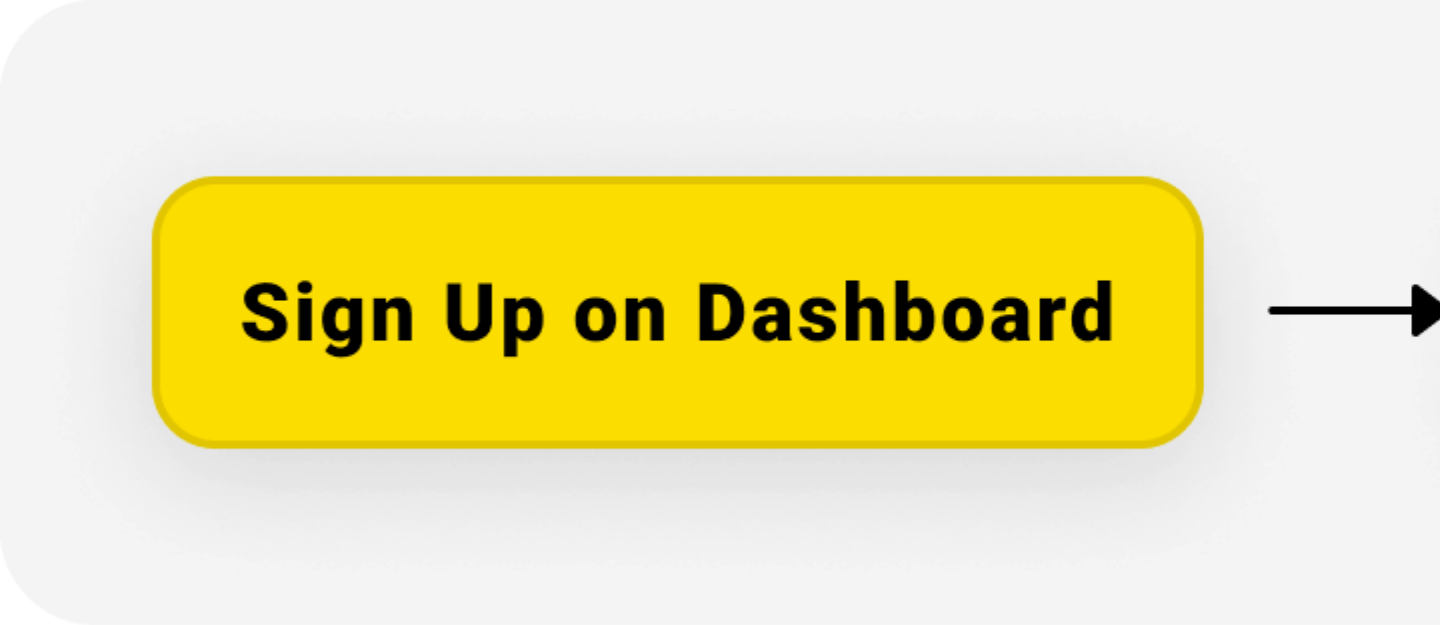
As an RIA, you can communicate with your customers directly. Alpaca will work hand-in-hand with you to market your service using Broker API in a compliant manner.

Currently, Alpaca does not support order allocation and advisory fee calculation as built-in functionality. These items are on our roadmap.

Integration Setup with Alpaca

If you are coming to Alpaca for the first time to build something using Broker API, please sign up for the dashboard. In this dashboard, you can acquire your API key for the sandbox environment and gain access to the test data immediately.

Suggest Edits



The sandbox behaves the same way as the live environment for the most parts with a few mocked. With this environment, you should be able to build a

complete demo app that you can show to your friends, investors and other community members. Going live from here would be a matter of testing and updating the API calls for the mocked endpoints.

To go live, we will onboard you for the business integration. For more details of this step, please refer to Going Live.

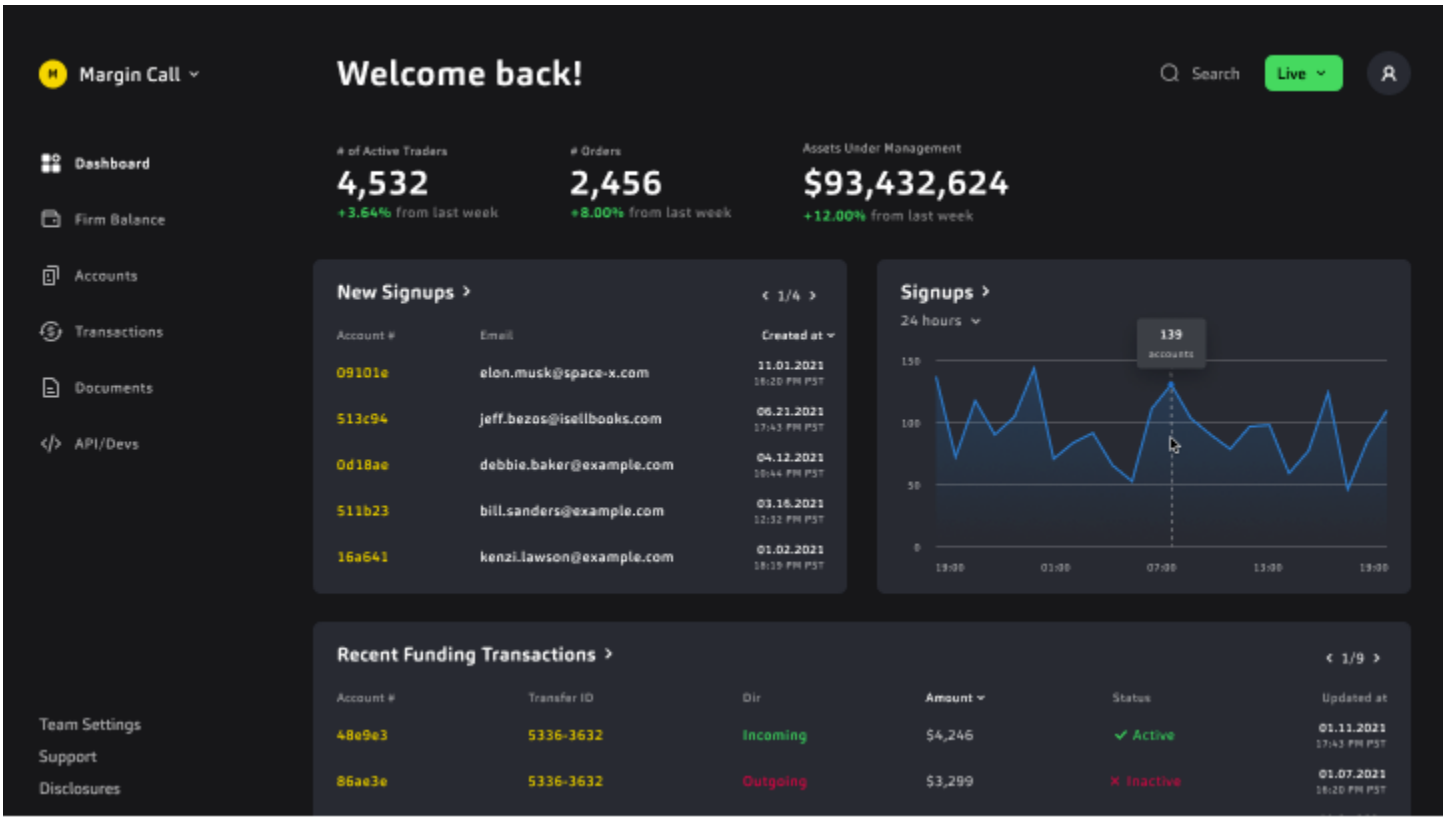
Once you go live, you can keep using the same dashboard to view customer activity and resolve issues for them for both Sandbox and Live. You will also get support for the broker operations and technology support based on the agreement.

Dashboard

Broker API users have access to a dashboard where you can view accounts and activities of your end users. You can sign up for an account with your email for free and get started with the sandbox API key.

You can invite team members to your account and view the same data as you make changes using API. You can switch the sandbox and live environment through this same dashboard. With all activity data, you can use this as your operation dashboard when going live as well.

You can assign different roles to each team member you invite.



Sandbox

You have full access to the sandbox while developing your integration for free of charge. The sandbox is built with the same code as live with a few different behaviors.

Account Approval

The account approval process slightly differs depending on your use case and you may need to test different scenarios in the sandbox first. The sandbox is fully automated with the account approval simulation with test fixtures, while the live environment may involve manual review and approval steps in some cases.

Trading

All trades that happen in the sandbox environment are simulated. The simulator engine is the same as our paper trading engine. All assumptions and mock logic follow the paper trading behavior. Please refer to the Trading API documentation.

Funding

The funding integration can vary depending on your country as well as the use case. In the sandbox environment, it is simplified with Transfer API. In order to simulate the deposit (credit) or withdrawal (debit) on the user account, simply call the POST method of Transfer API and it will become effective immediately. In the live environment, you may need to use Banks API as well as ACH endpoints if you are using ACH transfer within the USA. More details are described here.

Journal Approval

When you make a Journal API request, if the amount exceeds the pre-configured limit amount, it goes into a pending status. In the live environment, Alpaca's operation team is notified and manually reviews your request. In the sandbox environment, this process is simulated.

Firm Accounts

A firm account is an account owned by your business for the purpose of operations. We could support a variety of accounts based on your needs. Here are some basic ones.

- Deposit Account: This is a deposit clearing account required for all clients going live. The exact amount required is based on the number of accounts and the number of trades. This account balance moves rarely.
- Sweep Account: This is the main firm account you will be using to journal funds between your firm and your users. It can be used to simulate instant funding, to provide intraday credit and many other flexible funding strategies.
- Rewards Account: This account can be used to trigger rewards you want to set up on your app to fuel growth such as sign up rewards, referrals rewards and achievement based rewards. This account supports both cash and stock rewards.

You can view your Firm Account balance from the Broker Dashboard along with all activities associated with the account.

Going Live

Once you complete the sandbox integration, the next step is to go live. Please have another read about the differences between sandbox and live in the above section, and prepare for the go-live items. Generally speaking, we would need the following from you to open the live system for you.

- Your business entity documents, such as certificates of incorporation and tax ID
- Screenshots/video of your application interfaces
- KYC process document if you are fully-disclosed
- Expectation for the funding process if you already have something in mind

And with all this information provided, we will have a business agreement between you and Alpaca. We recommend allowing for enough time for the administrative tasks listed above, as we wouldn't want to delay your production launch unnecessarily.

When launching into production, we recommend to start from alpha/beta launch with a limited access, to ensure the operation works well, both on your side and Alpaca's side. Once the process is understood, you can go ahead with a full public launch. We are happy to consider participating in your PR / marketing launch!

Customer Account Opening

If you are a fully-disclosed broker-dealer, an RIA, or a trading app setup, you can open your end customer's account using Account API. The POST method allows you to submit all KYC information to Alpaca. There are slight differences between setups. #

Suggest Edits

Trading/Investing App and RIA

In this use case, Alpaca is responsible for the account approval step, while you can own the user experiences for collecting the end-customer information. We require you to collect a set of the information required for our approval process.

Upon the POST request, the account status starts from `SUBMITTED` status. Alpaca system will run the automatic KYC process asynchronously and update the KYC result as the account status. You can receive such updates in the Event API stream.

If all KYC information is verified without problems, the account status will be `APPROVED` and shortly transition to `ACTIVE`. In some cases, if the final approval is pending, the account status becomes `APPROVAL_PENDING` which will transition to `APPROVED` once it is approved. In the case of some action is required, the status becomes `ACTION_REQUIRED` and you will receive the reason for this. In most cases, you will need to collect additional information from the end user. One example would be that the residential address is not verified, so a copy of a document such as a utility bill needs to be uploaded. You can use Document API to upload additional documents when requested.

Fully-Disclosed Broker-Dealer

As a reminder, in this setup, you are required to have a proper broker-dealer license in your local jurisdiction and you are the broker on the record. Alpaca relies on your KYC process to open customers' accounts which you will send via the CIP API.

In this case, as soon as a `POST` request is made and all fields are validated, we will first screen the account against our internal list of blacklisted accounts and an exact, or similar, match against this list will result in the account moving to either `REJECTED` or `APPROVAL_PENDING`. If there is no match then the account status starts from `APPROVED` status, meaning you have approved the account opening. Therefore, you need to complete your KYC for the account before making the `POST` request.

Omnibus Broker-Dealer

In an omnibus setup, you will not request any new account opening. Your trading accounts will be set up by Alpaca when the go-live is approved. That said, you may want to simulate this structure using Account API and you can open as many accounts as you want in the sandbox environment even if you are an omnibus.

Account Type

Alpaca currently opens all accounts as margin accounts. We support individual taxable accounts and business accounts. Other types of accounts such as cash and IRA accounts are on our roadmap.

Even though all accounts at Alpaca are margin accounts, you have the ability to set accounts to be cash accounts (100% buying power) to disable margin trading for your users through account configurations here.

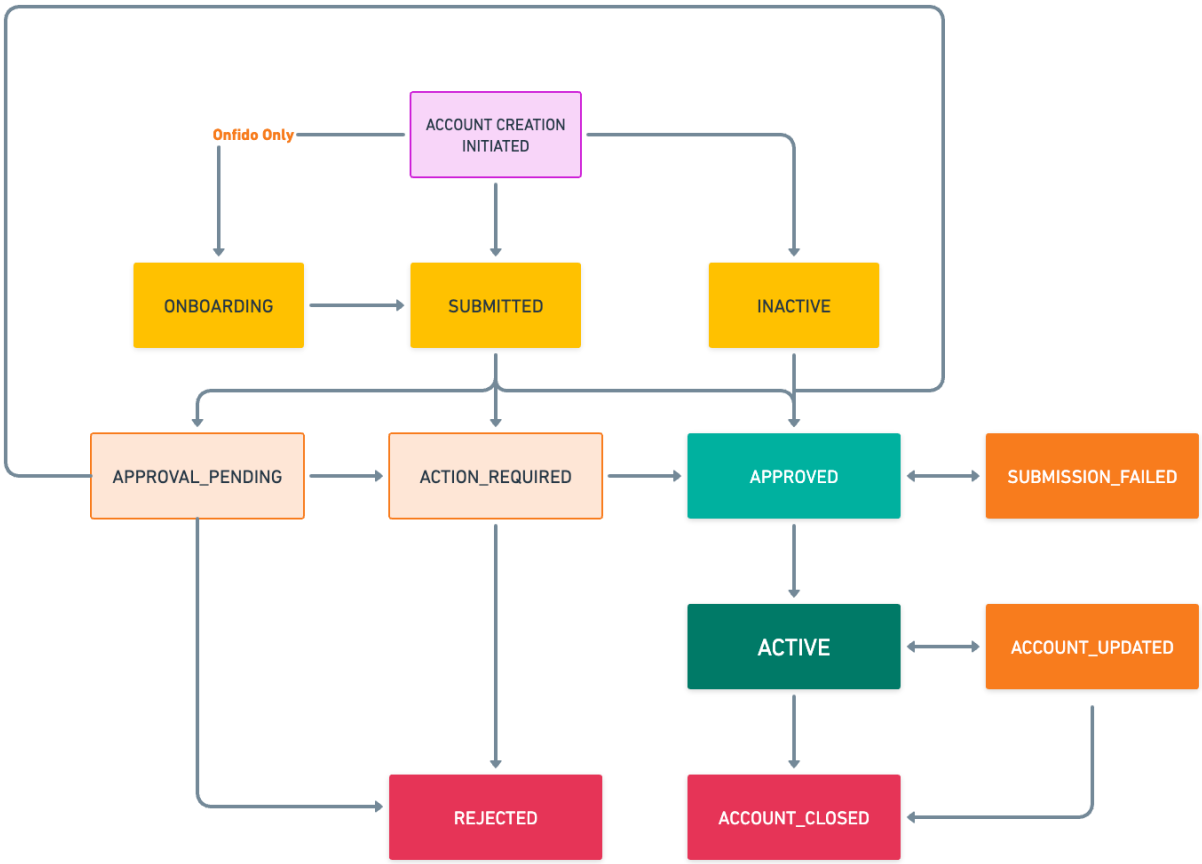
Accounts Statuses

Suggest Edits

The following are the possible account status values. Accounts will have both `status` and `crypto_status` with `status` denoting the account's equities trading status and `crypto_status` denoting the account's crypto trading status.
Most likely, the account status is `ACTIVE` unless there is an issue. The account status may get to `ACCOUNT_UPDATED` when personal information is being updated from the API, in which case the end user may not be allowed trading for a short period of time until the change is approved.
For more on creating an account check out our API reference section on the Accounts Endpoint.

status	description
`INACTIVE`	Account not set to trade given asset.
`ONBOARDING`	The account has been created but we haven't performed KYC yet. This is only used with Onfido.
`SUBMITTED`	The account application has been submitted and is being processed, this is a transitory status.
`SUBMISSION_FAILED`	The account failed to be created in Alpaca's system. Accounts in this status are resolved by Alpaca and no further action is needed.
`ACTION_REQUIRED`	The account application requires manual action and a document upload is required from the user. KYCResults contains information about the details.
`APPROVAL_PENDING`	The application requires manual checks from our team because the account did not pass the KYC automatic check, but most likely no document is required. KYCResults contains information about the details.
`APPROVED`	The account application has been approved, waiting to be ACTIVE, this is a transitory status.
`REJECTED`	The account application was rejected by our team. The account will not be able to continue to go active.

status	description
`ACTIVE`	The account is fully active and can start trading the enabled asset.
`ACCOUNT_UPDATED`	The account personal information is being updated which needs to be reviewed before being moved back to ACTIVE.
`ACCOUNT_CLOSED`	The account was closed, will not be able to trade or fund anymore.



Account statuses flowchart

W-8 BEN

For certain individuals, a W-8 BEN form should be submitted at onboarding. If the individual is not a registered U.S. taxpayer (not subject to a W-9), the W-8 BEN form may need to be submitted. The IRS explains here which individuals this applies to and provides instructions on completing the form. Every three years, in addition to the calendar year it was signed, a new W-8 BEN form must be submitted.

The form can be submitted in JSON, JSONC, PNG, JPEG or PDF. If submitting it in JSON, please see the W-8 BEN completed with the corresponding field names for the API here.

Note: The dates collected on the form are in a slightly different format than how they need to be submitted via Accounts API. It is requested by the user on the form in MM-DD-YYYY, but should be submitted as YYYY-MM-DD.

Domestic (USA) Accounts

Suggest Edits

State Validation

When creating or updating accounts with `country_of_tax_residence = USA` we will accept either the 2 letter abbreviation code for the state or the complete name of the state (case insensitive) as defined below:

State Abbreviation	State Name
AA	Armed Forces of the Americas
AE	Armed Forces of Europe
AK	Alaska
AL	Alabama
AP	Armed Forces of the Pacific
AR	Arkansas
AZ	Arizona
CA	California
CO	Colorado
CT	Connecticut
DC	District of Columbia
DE	Delaware
FL	Florida
GA	Georgia
HI	Hawaii
IA	Iowa
ID	Idaho

State Abbreviation	State Name
IL	Illinois
IN	Indiana
KS	Kansas
KY	Kentucky
LA	Louisiana
MA	Massachusetts
MD	Maryland
ME	Maine
MI	Michigan
MN	Minnesota
MO	Missouri
MS	Mississippi
MT	Montana
NC	North Carolina
ND	North Dakota
NE	Nebraska
NH	New Hampshire
NJ	New Jersey
NM	New Mexico
NV	Nevada
NY	New York
OH	Ohio
OK	Oklahoma
OR	Oregon
PA	Pennsylvania
RI	Rhode Island
SC	South Carolina

State Abbreviation	State Name
SD	South Dakota
TN	Tennessee
TX	Texas
UT	Utah
VA	Virginia
VT	Vermont
WA	Washington
WI	Wisconsin
WV	West Virginia
WY	Wyoming

Data Validations

Suggest Edits

As part of Alpaca Securities LLC's regulatory obligation to comply with new reporting requirements defined by FINRA, we are required to submit user information to comply with FINRA's Customer & Account Information System (CAIS). The CAIS system will begin validating the data for correct formatting so we need to ensure that data is provided in correct format at the time of account creation to avoid errors and potential delays with reporting.

This validation will be live in production on **March 25, 2024**. The validation will be released in sandbox first on **March 5, 2024** so you can carry out any testing required.

Validation Criteria

A validation check on user information submitted via the account creation (POST /v1/accounts) and update (PATCH /v1/accounts/{account_id}) endpoints will return a 422 error if the information submitted does not meet our validation criteria. The validation criteria will include the following:

- Name and Address Romanization
    - `given_name`, `middle_name`, `family_name`, `street_address`, `unit`, `city`, `state`, `postal_code`, `email_address`, and `tax_id` are all required to be provided in latin characters. The accepted input for these fields will be limited to ASCII character range 32-126

    - We have introduced the following fields to continue accepting name and address information in its original script if desired - `local_given_name`, `local_middle_name`, `local_family_name`, `local_street_address`, `local_unit`, `local_city`, and `local_state`
- `given_name` is now required for all users
- Tax ID Number Validation
    - `tax_id` is required for securities accounts
    - If the tax ID type is `USA_SSN` or `USA_TIN` then the following must be met:
        - No values having an Area Number (first three digits) of 000 nor 666.
        - No values having a Group Number (middle two digits) of 00.
        - No values having a Serial Number (last four digits) of 0000.
        - No values all of the same digit such as 000-00-0000, 111-11-1111, 333-33-3333, 666-66- 6666, 999-99-9999, nor all increasing or decreasing characters i.e. 123-45-6789 or 987-65-4321.
        - Values must be exactly 9 characters in length after dashes have been stripped
    - All tax ID types will undergo the following validation:
        - The length must be greater than 1 character i.e. submitting 0 as a tax ID will not be permitted
        - No values all of the same digit such as 000-00-0000, 111-11-1111, 333-33-3333, 666-66- 6666, 999-99-9999, nor all increasing or decreasing characters i.e. 123-45-6789 or 987-65-4321.
        - Max length of 40 characters
        - Only letters, digits, dashes (denoted by ASCII char 45), periods, and plus (+) signs will be permitted
        - Value most contain digits (i.e. submitting TIN_NOT_ISSUED or xxx-xxx-xxxx will not be permitted)
            - As a general reminder to our partners that onboard users in regions where tax ID numbers are not issued, there is still a requirement for a unique identifier to be submitted for those users. The identifier should be either a national identity card number, passport number, permanent resident

number, drivers license number, etc. We have introduced the following tax_id_type values to support these classifications. These are also available in our documentation here.
    - NATIONAL_ID
    - PASSPORT
    - PERMANENT_RESIDENT
    - DRIVER_LICENSE
    - OTHER_GOV_ID
- street_address Validation
    - No values consisting of only digits
    - Length must be greater than 1 character
- Postal Code Validation
    - If country of tax residence = USA:
        - The postal_code attribute will be required upon account creation
        - No values less than 5 characters in length and the first 5 characters must only contain digits
        - No values greater than 10 digits
- date_of_birth Validation
    - No values greater than or less than 10 characters in length. Values must be in YYYY-MM-DD format.
- Email addresses, after aliases are removed, are restricted to a maxim of 60 characters in length. Alpaca defines an alias as all characters after a + sign and before the @ sign.
- State Validation
    - For all countries
        - The max length for state should not be greater than 50 characters
        - State cannot consist of only digits. It can be alphanumeric
    - If country of tax residence = USA
        - State will be limited to either the 2 letter abbreviation code for the state or the complete name of the state as defined in our documentation here
- The city attribute cannot consist of only digits. It can be alphanumeric
- Whitespace validation
    - The following validation will be applied to the given_name, middle_name, family_name, street_address, unit, city, state, postal_code, email_address, tax_id_type, and tax_id fields on the Accounts API:

- The space character, denoted by ASCII character 32, will be the only whitespace character we accept
- Leading and trailing spaces present in the string will return a 422 error
- Additionally, we will be cleaning up the existing accounts in our system that contain invalid whitespace characters. We will follow up directly with the affected partners and share the complete list of accounts and data points that we will be updating.

Crypto Trading

- Alpaca supports crypto trading 24 hours a day, every day. Crypto is available for testing in sandbox, in case you want to allow your users to trade crypto please reach out to the Sales or Developer Success team.
- Suggest Edits
- 
- **To view the supported US regions for crypto trading, click here.**

Enabling Crypto for an Account

- To enable crypto trading for an account, the crypto agreements must be signed by the user. All account balances will represent the crypto trading activities.
- In the case of new users, the crypto agreement can be submitted via the Accounts API where `crypto_agreement` is part of the agreements attribute.
- *Part of the request*
- JSON

```
{
  "agreements": [
    {
      "agreement": "crypto_agreement",
      "signed_at": "2023-01-01T18:09:33Z"
    }
  ]
}
```

- In the case of existing users the account has to be updated with the `crypto_agreement` which can be submitted on the `PATCH /v1/accounts/{account_id}` endpoint.

- *Part of the request*
- JSON

```
{
  "agreements": [
    {
      "agreement": "crypto_agreement",
      "signed_at": "2023-01-01T18:13:44Z",
      "ip_address": "185.13.21.99"
    }
  ]
}
```

- Once the crypto agreement is added to the user account no further edits can be made to the agreements.
- To determine whether the account is all set to start trading crypto, use the `crypto_status` attribute from the Account API endpoint response object.
- *Sample Response*
- JSON

```
{
  "id": "9feee08f-22d2-4804-89c1-bf01166aad52",
  "account_number": "943690069",
  "status": "ACTIVE",
  "crypto_status": "ACTIVE"
}
```

Attribute	Description
INACTIVE	Account not enabled to trade crypto live
ACTIVE	Crypto account is active and can start trading
SUBMISSION_FAILED	Account submissions has failed

Supported Assets

- We have over 20 coins available to trade via our APIs. We constantly evaluate the list and aim to to grow the number of supported currencies.
- Tradable cryptocurrencies can be identified through the Assets API where the asset entity has `class = crypto` and `tradable = true`.

- JSON


```
{
  "id": "64bbff51-59d6-4b3c-9351-13ad85e3c752",
  "class": "crypto",
  "exchange": "CRXL",
  "symbol": "BTC/USD",
  "name": "Bitcoin",
  "status": "active",
  "tradable": true,
  "marginable": false,
  "shortable": false,
  "easy_to_borrow": false,
  "fractionable": true
}
```


- Please note that the symbol appears with `USD`, such as `BTC/USD` instead of `BTC`.
- 🚧
- ## Crypto Fee Revenue Notice
- If you enable non-USD crypto trading you will receive fees in the quote currency. Currently, non-USD quote crypto assets are BTC, USDC and USDT. As a broker business you would need to be ready to handle collecting crypto fees plus taking care of the necessary conversions if needed.
- Minimum Order Size
- The minimum quantity value that is accepted in an order. This value is calculated dynamically based on the selected notional equivalent minimum, based on the last close price of the relevant asset(s). The maximum decimal places accepted is 9 i.e 0.000000001 for all crypto assets.

- For `USD` pairs, the minimum order size calculation is: 1/USD asset price.
- For `BTC`, `ETH` and `USDT` pairs, the minimum order size is `0.000000002`.
- Min Trade Increment
- The minimum quantity allowed to be added to the `min_order_size`. E.g. if 0.1 we accept an order for 1.1 but we won't accept 0.9 because it's under the `min_order_size`. The maximum decimal places accepted are 9 i.e 0.000000001 for all crypto assets.
- Price Increment: The minimum notional value that is accepted in an order. Similar to Min Order Size but for notional orders. The maximum decimal places accepted are 9 i.e 0.000000001 for all crypto assets.

- All cryptocurrency assets are fractionable but the supported decimal points vary depending on the cryptocurrency.

Supported Orders

- When submitting crypto orders through the Orders API, Market, Limit and Stop Limit orders are supported while the supported `time_in_force` values are `gtc`, and `ioc`. We accept fractional orders as well with either `notional` or `qty` provided.
- Required Disclosures
- Below you will find required disclosure templates to safely support crypto in your applications as a broker with Alpaca.

- Onboarding Disclosures
- When onboarding your users as a broker offering crypto the following disclosure is required. During your onboarding flow make sure the user is able to read and affirmatively acknowledge, such as through a separate checkbox, the following text:

- I have read, understood, and agree to be bound by Alpaca Crypto LLC and [your legal entity] account terms, and all other terms, disclosures and disclaimers applicable to me, as referenced in the Alpaca Crypto Agreement. I also acknowledge that the Alpaca Crypto Agreement contains a pre-dispute arbitration clause in Section 26.

- Buy/Sell Order Screen Disclosures
- As a broker enabling the placement of cryptocurrency orders, the following disclosures should appear on the user's order entry screen, on the app or website, immediately prior to the user submitting the buy or sell order.

- *Buy Order Disclosure*
- By placing an order to buy [$ amount of / number of] [cryptocurrency], you are directing and authorizing Alpaca Securities LLC to transfer funds necessary to cover the purchase costs from your Alpaca Securities LLC account into your Alpaca Crypto LLC account. Cryptocurrency services are facilitated by Alpaca Crypto LLC. Cryptocurrencies are not securities and are not FDIC insured or protected by SIPC. Disclosures.

- *Sell Order Disclosure*
- By placing an order to sell [$ amount of / number of] [cryptocurrency], you are directing and authorizing Alpaca Crypto LLC to transfer settled funds from the sale into your Alpaca Securities LLC account.

Cryptocurrency services are facilitated by Alpaca Crypto LLC. Cryptocurrencies are not securities and are not FDIC insured or protected by SIPC. Disclosures.

- *Crypto Pairs Order Disclosure*
- By placing an order, you are directing and authorizing Alpaca Crypto LLC to exchange [X amount of Cryptocurrency] for [Y amount of cryptocurrency]. Cryptocurrency services are facilitated by Alpaca Crypto LLC. Cryptocurrencies are not securities and are not FDIC insured or protected by SIPC.

Margin and Short Selling

- Cryptocurrencies are non-marginable. This means that you cannot use leverage to buy them and orders are evaluated against `non_marginable_buying_power`.
- Cryptocurrencies are not shortable.

Trading Hours

- Crypto trading is offered for 24 hours everyday and your orders will be executed throughout the day.

Trading Limits

- Currently, an order (buy or sell) must not exceed $200k in notional. This is per an order.

Market Data

- Alpaca provides crypto data from multiple venues.
- Crypto data providers utilized by Alpaca:

Exchange	Exchange Code
`CBSE`	Coinbase
`CRXL`	Alpaca Crypto Exchange
`FLCX`	Falcon X

Funding Accounts

The funding process can vary depending on your setup and region and we support many cases, but everyone can do the same in the sandbox environment.
Suggest Edits

Sandbox Funding

In the sandbox environment, Transfer API simulates deposits and withdrawals to/from an account. The target account is immediately credited or debited upon such a request. Once an account is credited, the account can start trading with the Orders API.

ACH (US Domestic)

For US ACH, you will need to use Plaid to obtain the user's bank account information. You then pass the information to Alpaca using ACH API to create a Bank Link object. Once a Bank Link between a user and their bank account is established, then you can initiate both deposit and withdrawal transactions using the Transfers API.

Wire

Beginning June 1, 2022, we will begin charging for outgoing wires, both domestic and international. To help you provide the optimal customer experience we support two different flows for handling the fees:

1. **The end user pays the fee for every outgoing wire transfer that they initiate.** The `fee_payment_method` field will be equal to user in this case. It's important to note that the fee stated in your contract with Alpaca will automatically be deducted from the amount entered via the Transfers API so we strongly recommend adding a notice to your UI stating that the end user will incur a fee and they should incorporate that fee into their withdrawal request.
2. You will also have the option to **pay the fee on behalf of your user for any given transfer**. When creating the transfer, you will have to set the `fee_payment_method` field to invoice. The fee stated in your contract will not be deducted from the amount entered via the Transfers API but you will be charged this fee in your next monthly invoice.

Wire (US Domestic)

You can initiate a withdrawal transaction with wire transfer using the Transfers API. You need to create a bank object before that. For US domestic wire transactions, we will need ABA/routing number and the account number. You can supply additional text in each transaction.

In order for us to receive the deposits and book automatically, we need an "FFC" instruction in each incoming wire transaction. Please contact us for more details.

International Wire (SWIFT)

Alpaca supports international wire transfers and the API endpoint is the same as the US domestic case. You need to provide the SWIFT code and account number of the beneficiary, as well as the address and name of the receiving bank.

The FFC instructions above work for international wires too.

Cash Pooling

If you wish and are eligible, you can send customer deposits in a bulk to your firm account first and reconcile later using the Journals API.

We need to review the entire flow first to allow you to do so, and also you may need a local license to implement this process. Please check your counsel for the local requirements.

Travel Rule

In an effort to fight the criminal financial transactions, FinCEN enacted the Travel Rule that applies to fund transfers of more than $3,000. FAFT further adopted this from FinCEN to set the global standard, to regulate globally distributed crypto exchanges (virtual asset service providers; VASP) and many countries are due to implement this locally too.

Under this rule, financial institutions that transmit the funds are required to submit the following information to the recipient financial institutions (financial institutions here include banks and nonbanks; essentially any party that initiates the transfers).

- The name of the transmittor,
- The account number of the transmittor, if used,

- The address of the transmittor,
- The identity of the transmittor's financial institution,
- The amount of the transmittal order,
- The execution date of the transmittal order, and
- The identity of the recipient's financial institution
- The purpose of this requirement is for the investigators to track the flow of funds in case they need to. Failure to do so could cause a civil enforcement.

When you use Journal API to bundle a bulk of transfers for the end-users, you will need to tell about the breakdown and each transmitter information using the optional fields of the POST request.

Alpaca retains the collected information for at least five years. If the journal activities are used as part of the money transfer (other than cash movement within Alpaca), and if the journal requests don't contain the transmitter information, we may contact you.

Instant Deposit (Beta)

As international money transfers can take days usually, under certain conditions, Alpaca supports instant deposit for better user experience. Please contact us for more details.

Post-trade Settlement

We support the post-trade settlement process (higher requirements and restrictions apply). Please contact us for more details.

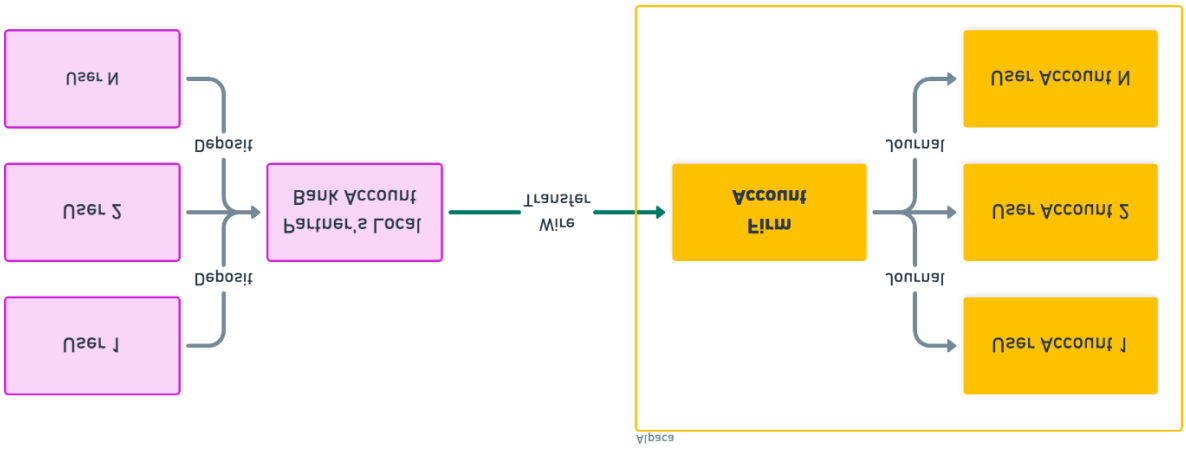
Journals API

Suggest Edits

Journals API allows you to move cash or securities from one account to another.

For more on creating and retrieving journals please check out our API reference section on journals.

The most common use case is cash pooling, a funding model where you can send bulk wires into your firm account and then move the money into each individual user account.



Cash pooling funds flow

There are two types of journals:

JNLC
Journal cash between accounts. You can simulate instant funding in both sandbox and production by journaling funds between your pre-funded sweep accounts and a user's account.

You can only journal cash from a firm account to a user account and vice-versa but not from customer to customer.

JNLS
Journal securities between accounts. Reward your users upon signing up or referring others by journaling small quantities of shares into their portfolios.

You can only journal securities from a firm account to a user account and not vice-versa or customer-to-customer.

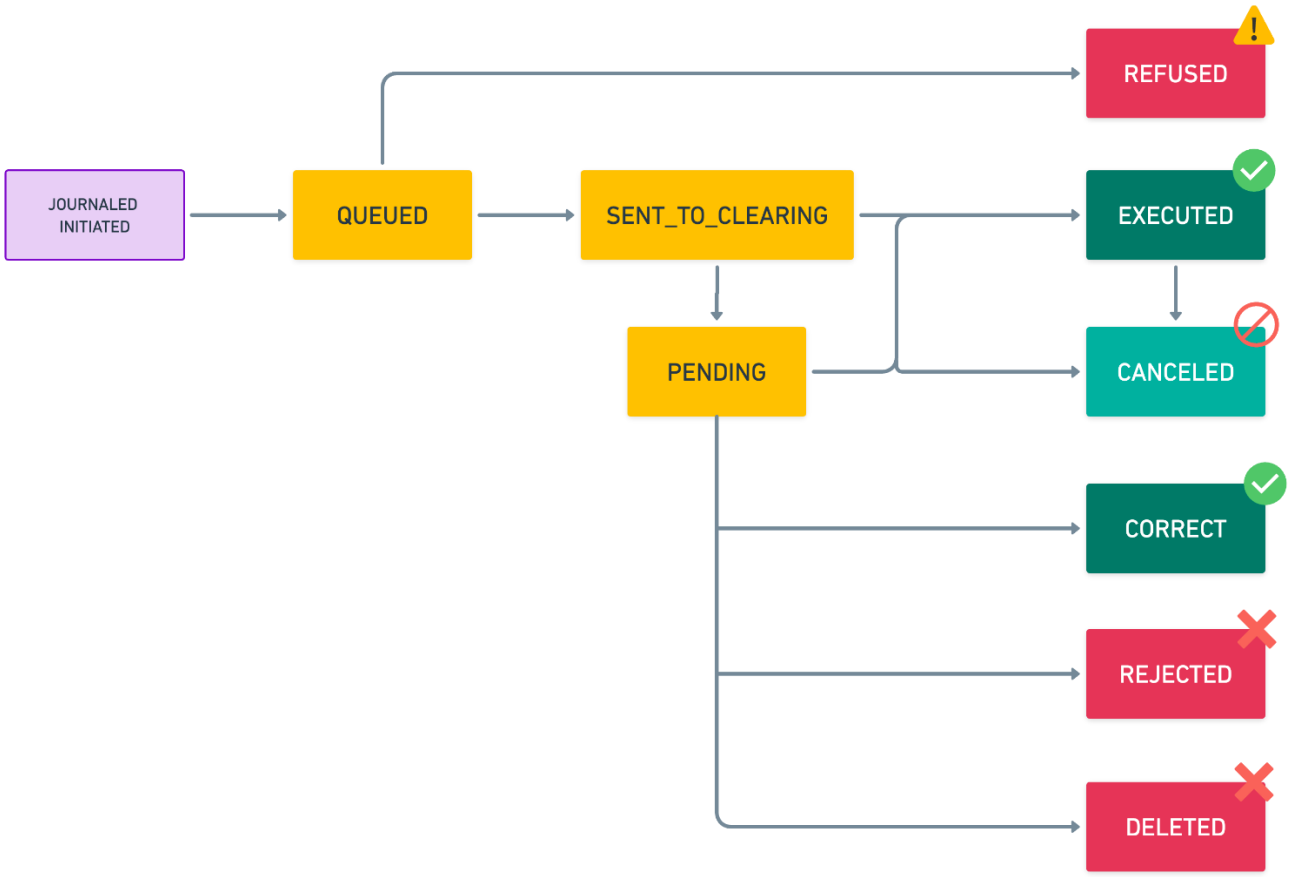
Journals Status

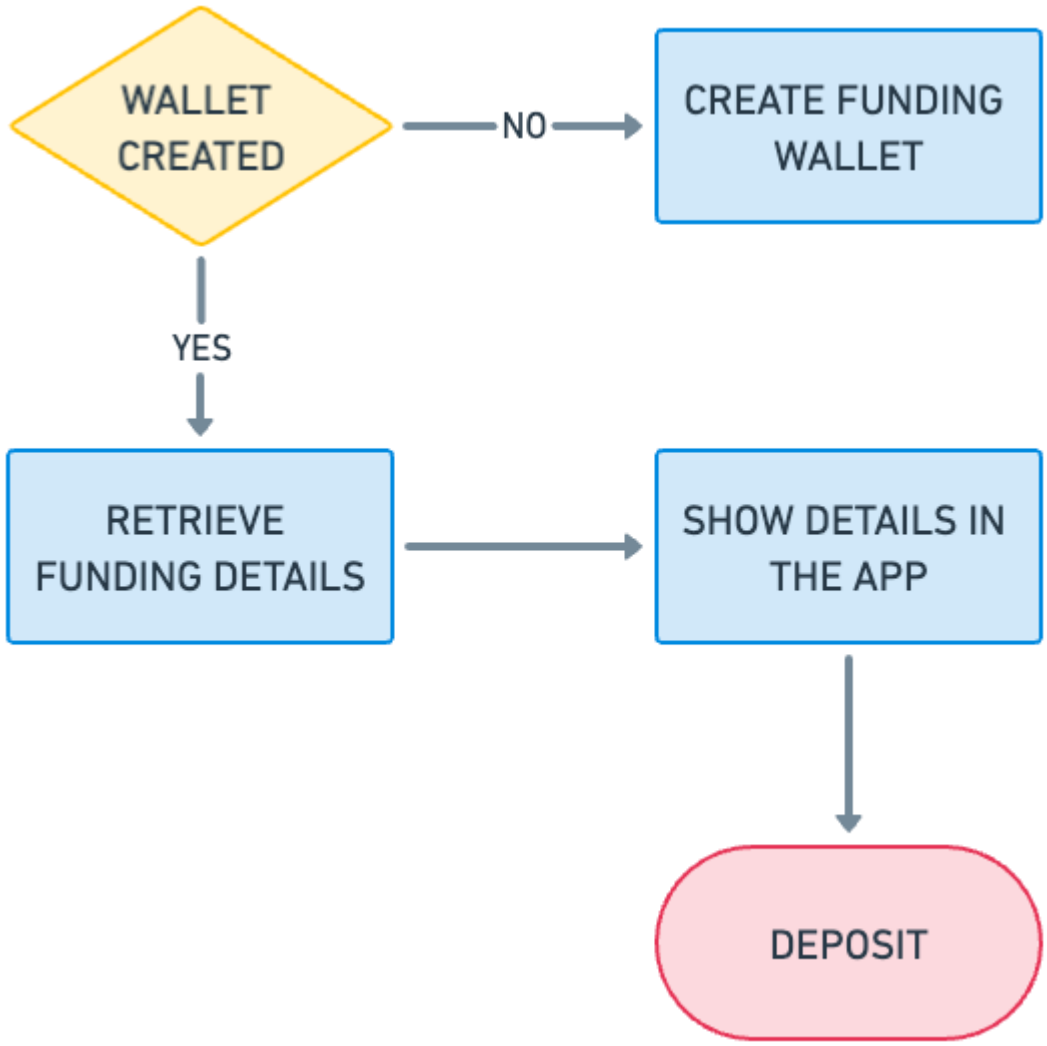
The most common status flow for journals is quite simple:

1. Upon creation, the journal will be created in a `queued` state.
2. Then, the journal will be `sent_to_clearing` meaning that the request has been submitted to our books and records system.
3. Lastly, if there are no issues the journal will be `executed`, meaning that the cash or securities have been successfully moved into the receiving account.

Still, there are other cases in which the journal is `rejected`, `refused` or requires manual intervention from Alpaca's cashiering team.

Status	Description
`queued`	This is the initial status when the journal is still in the queue to be processed.
`sent_to_clearing`	The journal has been sent to be processed by Alpaca’s booking system.
`executed`	The journal has been completed and the balances have been updated for the accounts in the transaction. In some rare cases, journals can be reversed from this status by Alpaca's cashiering team if the transaction is not permitted.
`pending`	The journal is pending to be processed as it requires manual approval from Alpaca oper example, this can be caused by hitting the journal limits.
`rejected`	The journal has been manually rejected.
`canceled`	The journal has been canceled, either via an API request or by Alpaca's operations team
`refused`	The journal was never posted in Alpaca's ledger, probably because some of the prelimin checks failed. A common example would be a replayed request in close succession, wh first request is executed and the second request fails the balance check.
`correct`	The journal has been manually corrected. The previously executed journal is cancelled journal with the correct amount is created.
`deleted`	The journal has been deleted from our ledger system.


JOURNALED
INITIATED
QUEUED
SENT_TO_CLEARING
PENDING
REFUSED
EXECUTED
CANCELED
CORRECT
REJECTED
DELETED



ACH Funding

Suggest Edits

Plaid Integration for Bank Transfers

We have integrated with Plaid to allow you to seamlessly link your Plaid account to Alpaca. The integration will allow your end-users to verify their account instantly through Plaid's trusted front-end module.

Leveraging this allows you to generate Plaid Processor Tokens on behalf of your end-users, which allows Alpaca to immediately retrieve a user's bank details in order to deposit or withdraw funds on the Alpaca platform.

You can utilize your Plaid account and activate the Alpaca integration within the Plaid dashboard.

The integration requires Plaid API Keys

Obtaining a Plaid Processor Token

A Plaid processor token is used to enable Plaid integrations with partners. After a customer connects their bank using Plaid Link, a processor token can be generated at any time. Please refer to the Plaid Processor Token using Alpaca page for creating a token and additional details.

Exchange token

```
curl -X POST <https://sandbox.plaid.com/item/public_token/exchange>
  -H 'Content-Type: application/json'
  -d '{
    "client_id": "PLAID_CLIENT_ID",
    "secret": "PLAID_SECRET",
    "public_token": "PUBLIC_TOKEN"
  }
```

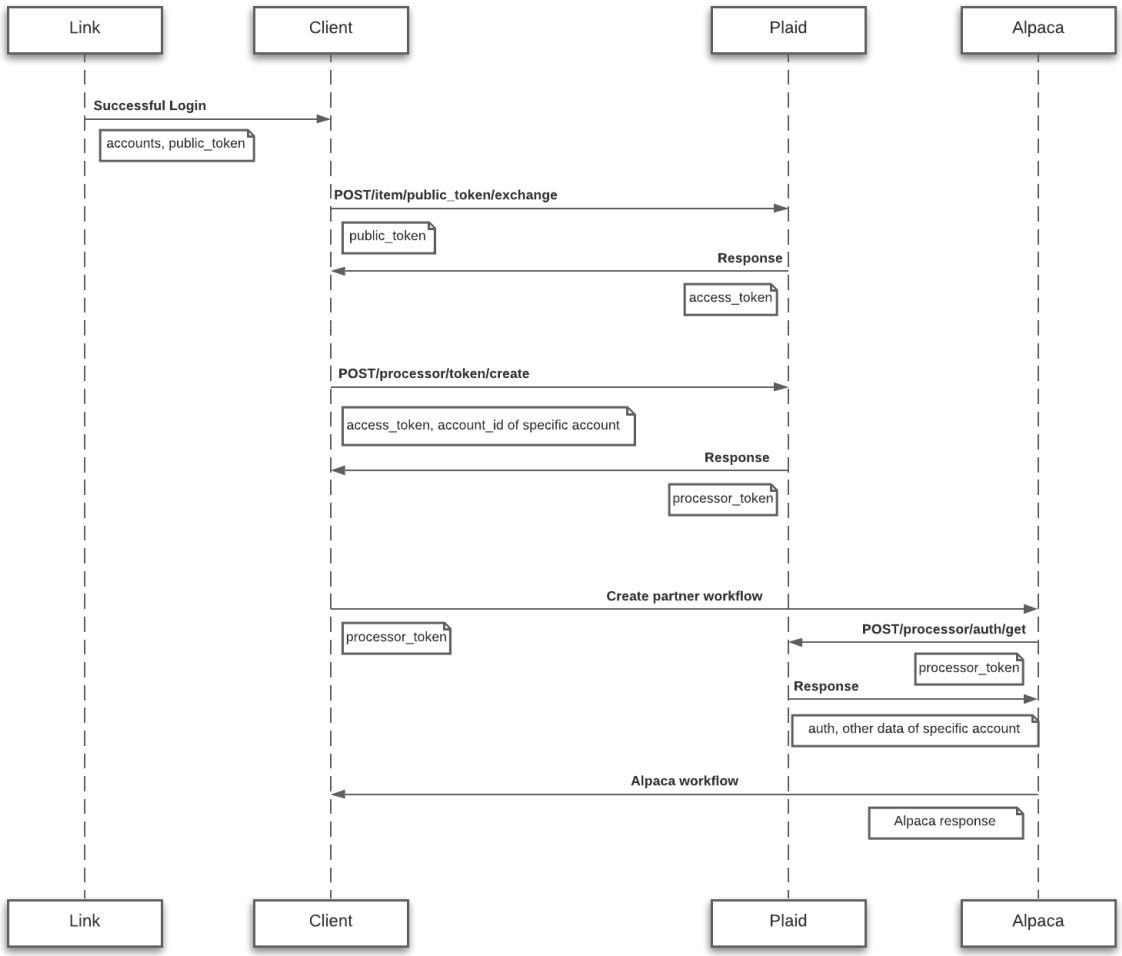
Create a processor token for a specific account id.

```
curl -X POST <https://sandbox.plaid.com/processor/token/create>
-H 'Content-Type: application/json'
-d '{
  "client_id": "PLAID_CLIENT_ID",
  "secret": "PLAID_SECRET",
  "access_token": "ACCESS_TOKEN",
  "account_id": "ACCOUNT_ID",
  "processor": "alpaca"
}
```

For a valid request, the API will return a JSON response similar to:

```
{
  "processor_token": "processor-sandbox-0asd1-a92nc",
  "request_id": "m8MDnv9okwxFNBV"
}
```

Processor Token Flow



1. End-user links bank account using Plaid.
2. Plaid returns a public token to you.
3. You will submit a public token to Plaid in exchange for an access token.
4. You will submit access token to Plaid's /processor/token/create endpoint and receive Processor Token (specific to Alpaca).
5. You will make a call to the processor endpoint to pass Alpaca the processor token, to initiate the payment. To pass the processor token use the ACH relationships endpoint (Link).

Sample Request

```
POST /v1/accounts/{account_id}/ach_relationships
```

text

```
{
    "processor_token": "processor-sandbox-161c86dd-d470-47e9-a741-d381c2b2cb6f"
}
```

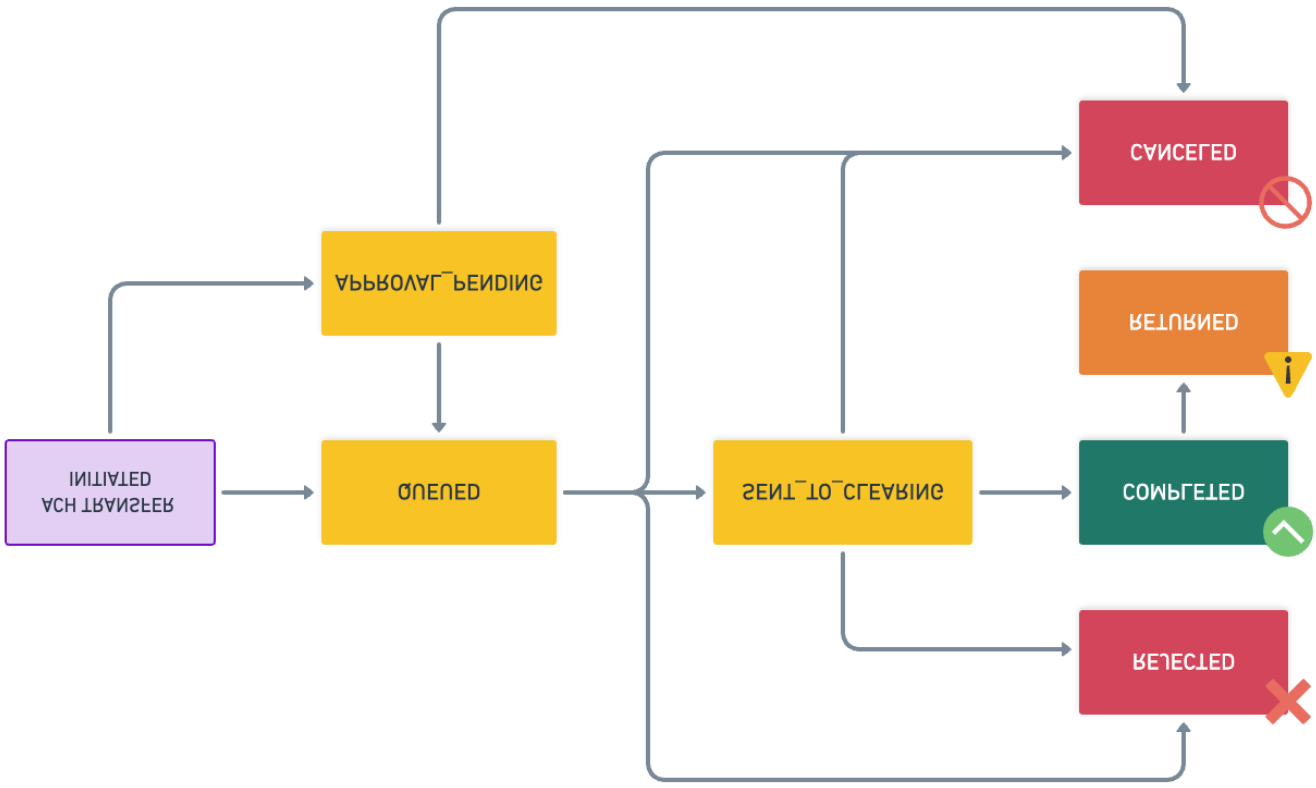
Sample response


```
{
  "id": "794c3c51-71a8-4186-b5d0-247b6fb4045e",
  "account_id": "9d587d7a-7b2c-494f-8ad8-5796bfca0866",
  "created_at": "2021-04-08T23:01:53.35743328Z",
  "updated_at": "2021-04-08T23:01:53.35743328Z",
  "status": "QUEUED",
  "account_owner_name": "John Doe",
  "nickname": "Bank of America Checking"
}
```


6. Alpaca makes a call to Plaid to retrieve the Account and Routing number* using the processor token.
7. Alpaca saves the processor token and account and routing number internally for future use. Alpaca uses account information for NACHA file creation and processing.

*Can include Auth, Identity, Balance info - if the broker API wants to initiate a transfer, we use the transfer endpoint.

ACH Status Flow



Trading

All the functionality from Alpaca Trading API for direct users is supported under Broker API. Please read the documentation for more details. There are additional capabilities for Trading API in the broker setup.
Suggest Edits

Fractional Shares

Under the correspondent authentication, Trading API is extended for fractional shares trading.

Asset API

Asset entity has an extra field named fractionable with type boolean, it is set to true if the asset is marked for fractional trading. Check out more about our Assets API here.

Order API

The `POST` method (to submit order) has an extra parameter called `notional` to denote the dollar amount to buy. This parameter is mutually exclusive to the qty parameter. You can also specify the `qty` parameter with fractional quantity. Note that the fractional shares trading is currently supported only for market day orders and any other type of orders will result in API error. You can submit such fractional share orders in the night to execute at the next market open.
The order entity includes the `notional` value if the order was submitted with the notional value. In this case, the order entity omits the `qty` field.

Account Configuration API

Account configuration API adds another configuration key called `fractional_trading` and defaults to `true`. If you want to disable fractional trading for a specific account for any reason, you can set this to `false`.

Commissions

You have the option to charge the commission for each order. You will need to contact Alpaca first to set up the commission structure, but once it's set up, you can submit customer orders with a `commission` parameter indicating the dollar amount to charge. The respective field is attached in the order entity in the API response.

Order Sub-tagging (Omnibus)

If you are an omnibus setup, we ask you to submit a “sub-tag” value in each order. This is for us to understand the order flow better from the trade surveillance requirements. In case you fail to attach proper sub-tags, we may need to reject all of the order flows coming from you as we may not be able to segregate particular malicious activities.

Portfolio Rebalancing

Suggest Edits

Rebalancing API offers investment advisors a way to easily create investment portfolios that are automatically updated to the specified cash, stock symbol percentage weights, rebalance conditions, and triggers selected. Some helpful definitions before an overview of the rebalancing flow:

- Portfolios: An allocation containing securities and/or cash with specific weights and conditions to be met
- Subscriptions: Accounts can be subscribed to a created portfolio and follow rebalancing events to ensure the account if kept in sync with the target portfolio
- Runs: A run is a set of orders that will be sent for execution to achieve a goal (liquidating a specified amount to set it aside for withdrawal or doing a full rebalance to the target allocation)

Rebalancing API Resource

- Postman Collection
- How to Get Started with Rebalancing API
- Rebalancing API Reference

Types of Rebalancing

Rebalancing API offers two types of rebalancing conditions:

- **Drift Band**: When a portfolio breaches a certain threshold, irrespective of the time period elapsed, the portfolio is adjusted. For

instance, if we put a +/- 10% band on a portfolio, we would automatically adjust the entire portfolio when we reach the threshold for one of the holdings

- **Calendar**: At the desired period, the state of the portfolio is analyzed and the portfolio is rebalanced to the default portfolio. For example, on April 1st our 50:50 AAPL TLT portfolio is not 55:45, so we would need to liquidate TLT and buy more AAPL to return to the desired state of exposure of 50:50.

Create a Portfolio

To create a portfolio use the Create Portfolio `POST` endpoint.
See below see example payload to create a portfolio with a mix of cash and securities:

JSON

```
{
    "name": "Balanced",
    "description": "A balanced portfolio of stocks and bonds",
    "weights": [
        {
            "type": "cash",
            "percent": "5"
        },
        {
            "type": "asset",
            "symbol": "SPY",
            "percent": "60"
        },
        {
            "type": "asset",
            "symbol": "TLT",
            "percent": "35"
        }
    ],
    "cooldown_days": 7,
    "rebalance_conditions": [
        {
            "type": "drift_band",
            "sub_type": "absolute",
            "percent": "5"
        },
        {
            "type": "drift_band",
            "sub_type": "relative",
            "percent": "20"
        }
    ]
}
```

Once exeuted, you can find the portfolio ID in the response payload similar to the one below. In our case our newly created portfolio ID is 2d49d00e-ab1c-4014-89d8-70c5f64df2fc. This will be needed to be able to subscribe an account to follow this new portfolio.

JSON


```
{
    "id": "2d49d00e-ab1c-4014-89d8-70c5f64df2fc",
    "name": "Balanced Two",
    "description": "A balanced portfolio of stocks and bonds",
    "status": "active",
    "cooldown_days": 7,
    "created_at": "2022-08-07T14:56:45.116867815-04:00",
    "updated_at": "2022-08-07T14:56:45.196857944-04:00",
    "weights": [
        {
            "type": "cash",
            "symbol": null,
            "percent": "5"
        },
        {
            "type": "asset",
            "symbol": "SPY",
            "percent": "60"
        },
        {
            "type": "asset",
            "symbol": "TLT",
            "percent": "35"
        }
    ],
    "rebalance_conditions": [
        {
            "type": "drift_band",
            "sub_type": "absolute",
            "percent": "5",
            "day": null
        },
        {
            "type": "drift_band",
            "sub_type": "relative",
            "percent": "20",
            "day": null
        }
    ]
}
```


You can also list all your created portfolios with the List All Portfolios endpoint.

Subscribe Account to a Portfolio

Once you have a portfolio created, the next step is to subscribe a given account to follow a portfolio. This will ensure that when rebalancing conditions are found the accounts is subscribed to have the needed orders executed.

To subscribe an account to our newly created portfolio and its rebalancing conditions we create a new subscription. For example, to subscribe account ID bf2b0f93-f296-4276-a9cf-288586cf4fb7 to our newly created portfolio from before, we use the Create Subscriptions endpoint with the following JSON payload,

JSON

```
{
    "account_id": "bf2b0f93-f296-4276-a9cf-288586cf4fb7",
    "portfolio_id": "57d4ec79-9658-4916-9eb1-7c672be97e3e"
}
```

Check Rebalancing Events

Once an account is subscribed to a portfolio, we need to wait for the first rebalancing event to happen. We can check completed rebalancing events for all our accounts by using the List All Runs endpoint.

cURL

```
curl --location --request GET
'{{HOST}}/v1/beta/rebalancing/runs?status=COMPLETED_SUCCESS' \
--header 'Authorization: Basic <TOKEN>' \
--data-raw ''
```

See example payload of a succesful run,

JSON


```
{
    "runs": [
        {
            "id": "36699e7f-56a0-4b87-8e03-968363f4b6df",
            "type": "full_rebalance",
            "amount": null,
            "initiated_from": "system",
            "status": "COMPLETED_SUCCESS",
            "reason": null,
            "account_id": "b3130eeb-1219-46f3-8bfb-7715f00d736b",
            "portfolio_id": "4ad7d634-a60d-4e6e-955f-3c68ee24d285",
            "weights": [
                {
                    "type": "cash",
                    "symbol": null,
                    "percent": "5"
                },
                {
                    "type": "asset",
                    "symbol": "SPY",
```

```
            "percent": "60"
        },
        {
            "type": "asset",
            "symbol": "TLT",
            "percent": "35"
        }
    ],
    "orders": [
        {
            "id": "c29dd94b-eaaf-4681-9d1f-4fd47571804b",
            "client_order_id": "cb2d1ff5-8355-4c92-84d7-dfff43f44cb2",
            "created_at": "2022-03-08T16:51:07.442125Z",
            "updated_at": "2022-03-08T16:51:07.525039Z",
            "submitted_at": "2022-03-08T16:51:07.438495Z",
            "filled_at": "2022-03-08T16:51:07.520169Z",
            "expired_at": null,
            "canceled_at": null,
            "failed_at": null,
            "replaced_at": null,
            "replaced_by": null,
            "replaces": null,
            "asset_id": "3b64361a-1960-421a-9464-a484544193df",
            "symbol": "SPY",
            "asset_class": "us_equity",
            "notional": "30443.177578017",
            "qty": null,
            "filled_qty": "72.865432211",
            "filled_avg_price": "417.8",
            "order_class": "",
            "order_type": "market",
            "type": "market",
            "side": "buy",
            "time_in_force": "day",
            "limit_price": null,
            "stop_price": null,
            "status": "filled",
            "extended_hours": false,
            "legs": null,
            "trail_percent": null,
            "trail_price": null,
            "hwm": null,
            "subtag": null,
            "source": null
        },
        {
            "id": "ab772dcb-b67c-4173-a5b5-e31b9ad236b5",
            "client_order_id": "d6278f6c-3010-45ce-aaee-6e64136deec0",
            "created_at": "2022-03-08T16:51:07.883352Z",
            "updated_at": "2022-03-08T16:51:07.934602Z",
            "submitted_at": "2022-03-08T16:51:07.877726Z",
            "filled_at": "2022-03-08T16:51:07.928907Z",
            "expired_at": null,
            "canceled_at": null,
            "failed_at": null,
            "replaced_at": null,
            "replaced_by": null,
            "replaces": null,
            "asset_id": "a106d0ef-e6f2-4736-8750-5dee1cadf75b",
            "symbol": "TLT",
```

```
                    "asset_class": "us_equity",
                    "notional": "17121.076868834",
                    "qty": null,
                    "filled_qty": "124.408348124",
                    "filled_avg_price": "137.62",
                    "order_class": "",
                    "order_type": "market",
                    "type": "market",
                    "side": "buy",
                    "time_in_force": "day",
                    "limit_price": null,
                    "stop_price": null,
                    "status": "filled",
                    "extended_hours": false,
                    "legs": null,
                    "trail_percent": null,
                    "trail_price": null,
                    "hwm": null,
                    "subtag": null,
                    "source": null
                }
            ],
            "completed_at": null,
            "canceled_at": null,
            "created_at": "2022-03-08T16:36:07.053482Z",
            "updated_at": "2022-03-08T16:51:08.53806Z"
        },
        ...
    ],
    "next_page_token": 100
}
```


Note

Cash inflows to the account (deposits, cash journals, etc.) will trigger buy trades to reduce drift.

Manually Trigger Rebalancing Event (Run)

Rebalancing API will automatically configure systems to watch for portfolio rebalancing conditions and execute necessary orders. However, if you need to execute a rebalancing run see reference of Create Run endpoint.

Manually executing a run is currently only allowed for accounts who do not have an active subscription.

FAQ

Q: What is the minimum cash required for a rebalancing run?

For a rebalancing run to occur, there must be a minimum of $1 per asset in the portfolio.

Example 1 - A portfolio that is 50% AAPL, 25% TSLA, 25% GOOGL
Assuming all other rebalancing conditions and cooldown days are met, a rebalancing run would only occur with at least $1 per asset in the portfolio deposited. Anything less would not trigger a rebalance.

Example 2 - A portfolio that is 50% AAPL, 25% TSLA, 25% CASH
Assuming all other rebalancing conditions and cooldown days are met, a rebalancing run would only occur with at least $1 per asset in the portfolio deposited. Anything less would not trigger a rebalance.

For an invest_cash run to occur, a minimum of $10 must be deposited.

Q: Can rebalancing run partially for some assets where minimum notional is satisfied but not run for the assets where minimum notional is not satisfied?

Yes. If a portfolio has 11 assets, but only $10 is deposited, there will be a partial rebalancing where 10 assets will be invested in.

Q: When does a Rebalancing Job or invest_cash job run at Alpaca?

When cash is first invested into a portfolio (assuming the minimum cash requirement is met), a rebalancing job of type full_rebalance will occur.

Thereafter, if a minimum of $10 is invested, an invest_cash job will run regardless of the portfolio rebalancing conditions and the cooldown period.

After the cooldown period, and if the rebalancing conditions are met, then a rebalancing job will run.

Both the rebalance and invest_cash jobs run between 930am -330pm EST .

Q: What are the steps for Cash Withdrawal if a user wants to close the positions and withdraw their cash?

The steps are as follows:

1. Unsubscribe the user using the https://broker-api.sandbox.alpaca.markets/v1/rebalancing/subscriptions/{subscription_id} DELETE request.
2. Execute a manual run with revised portfolio weights using the https://broker-api.sandbox.alpaca.markets/v1/rebalancing/runs POST request.
3. Wait for the desired amount to be available as withdrawable cash, and then make a GET request to https://broker-api.sandbox.alpaca.markets/v1/trading/accounts/{account_id}/account and check if cash_withdrawable has the desired amount.

*Note: The waiting time is currently T+2 days, and will be T+1 days post May 2024.

4. Journal the money from user account to firm account using the https://broker-api.sandbox.alpaca.markets/v1/journals POST request.
5. Subscribe the user back to their portfolio using the https://broker-api.sandbox.alpaca.markets/v1/rebalancing/subscriptions POST request.

6.Daily Processes and Reconcilations

7. Suggest Edits

8. Daily Processes

9. There are a few daily timings you want to keep in mind when you think about the operation.

Process	Timing	Notes
Beginning-of-day Sync (BOD)	02:15AM-02:30AM EST	Trading accounts are updated with the previous day end-of-day values. confirms are also synchronized around this time.
Incoming wire processing	08:00AM-08:30AM EST	The incoming wires with FFC instructions are booked
Outgoing wire cutoff	04:00PM EST	The outgoing wire requests before the cutoff will be processed for the
ACH cutoff	02:00PM EST	The credit/debit of ACH requests before the cutoff will be processed fo

Process	Timing	Notes
Trade reporting	06:30PM-07:30PM EST	The day's trades are finalized and reported.
End-of-day calculation (EOD)	11:00PM-11:30PM EST	Close the day's book, mark to market positions, cost basis calculation, r requirements calculation etc.

10. Mandatory Corporate Actions

11. Currently the corporate actions are processed a semi-automated way, and you will see such records in Activity API as they happen. We are working to provide upfront information separately in the future.

12. Dividends

13. Dividends are the most common corporate actions. The cash is paid (credited) to the customer accounts after the pay date, as we receive the cash from DTC. Please note that the actual credit transactions may be after the pay date if we don't receive the cash from DTC. When such payout is transacted, you will see the account activity in Activity API as the DIV entry type.

14. Dividends are income gain. If your end customers are non-US residents, 30% withholding is applied by default. In case you claim to apply different rates for the tax treaty, please contact us.

15. Dividends are processed without waiting for DTC in the sandbox environment. This may not reflect the live side operation.

16. Forward Splits and Reverse Splits

17. Share splits are processed as they happen and the beginning-of-day process will update the positions of the customer accounts. Both appear as a SPLIT entry type in the activity. In the case of reverse splits, there might be the cash in lieu for non-divisible shares which will not be processed immediately until we receive the cash from DTC.

18. Symbol/CUSIP Change and Listing/Delisting

19. The symbol or CUSIP can change one day for a particular asset. The asset master data is refreshed on a daily basis and we do recommend you retrieve the asset endpoint every morning before the market open (or

after the beginning-of-day timing). While Alpaca does not currently participate in the initial public offering, such stock on the IPO day will become tradable on the day it is listed, and start filling orders once the secondary market opens.

20. Other Events

21. Mergers, acquisitions, and other type events are processed manually in our back office as they are rare and each case is often unique. Please contact Alpaca's broker-dealer operation team if you have any questions.

22. ACATS

23. Alpaca processes both sending and receiving ACATS requests. As of today, you can request our operation team for the receiving request, but we plan to provide this service as an API in the future.

24. Monthly Processes

25. Monthly statement emails should be sent for the prior month on or before the 10th of the following month - for example, for the monthly statement for August, delivery via email must be on or prior to September 10.

26. Statements and Confirms

27. Suggest Edits

28. Requirements

29. Under the FINRA and SEC rules, Alpaca is required to ensure the customer statements and trade confirms are delivered correctly in time to the end customers. That being said, the actual communication and delivery do not have to be done by Alpaca directly. Very often, you want to own the full user experiences and to be responsible for these communications, which is totally possible.

30. Document API Integration

31. You can retrieve the generated reports in PDF format through the Documents API. You can store the files on your storage if it is required for your regulation purpose, or you can let your customers download the files using the URL returned in the API response. If you are a fully-disclosed broker-dealer, you can insert your firm logo, name and address in the PDF template. Please send those data to Alpaca.

32. If you need even more customization on the template, we are currently working on the new API endpoint which will return only the data points so

that you can build fully-customized documents with your own template. Alpaca still needs to review your final version of customized documents before delivering to the end customers for the first time.

Local Currency Trading (LCT)

Local Currency Trading allows customers to trade US equities in over 15+ local currencies, with FX conversion done on-the-fly. Customers can place, monitor and sell their positions in their local currency.

Suggest Edits

API responses are all in your local currency, with all calculations handled by Alpaca.

Further below, we will examine the some common scenarios with LCT. The recurring theme you will notice is that many of Alpaca's API commands are almost the same, it is just the response that have changed. In some cases, barring the introduction of a currency specification or a swap rate, the only indication of the trade being in Local Currency is the inclusion of a USD second order JSON.

For further questions about LCT, such as supported currencies or any other relevant details, see LCT FAQs.

Supported Features

Features	LCT	Broker AP
Allows trading in user's/broker local currency of US equities	✅	⛔
Supports JIT	✅	✅
Stop and Limit orders	⛔	✅
Swap rate on the orders endpoint	✅	⛔
Supports crypto trading	⛔	✅
Market Data	✅ (in local currency)	⛔
Omnibus	✅	✅
Omnibus in subledger	✅	✅
Fully-disclosed account type	✅	✅
SSE Events	✅	✅

Features	LCT	Broker AP
Rebalancing	⛔	✅
Margin Trading	⛔	✅

Get Market Data

With LCT, we have introduced a currency parameter for stock market data. You can request pricing data for any equity and we will handle the necessary conversions to quote the asset in the requested local currency.

The example below shows how to get pricing data for AAPL in Euro. The pricing information is converted from USD to the relevant local currency on the fly with the latest FX rate at the point in time of query.

cURL

```
curl --request GET 'https://data.alpaca.markets/v2/stocks/AAPL/bars?start=2021-05-01T0:00:00Z&end=2021-05-31T11:00:00Z&timeframe=1Min&currency=EUR'
```

JSON

```
{
    "bars": [
        {
            "t": "2021-05-03T08:00:00Z",
            "o": 109.58,
            "h": 109.87,
            "l": 109.58,
            "c": 109.69,
            "v": 4106,
            "n": 75,
            "vw": 109.72
        },
        ...
    ],
    "currency": "EUR",
    "symbol": "AAPL",
    "next_page_token": "QUFQTHxNfDIwMjEtMDUtMDRUMTI6NTA6MDAuMDAwMDAwMDAwWg=="
}
```

Note currency key value is `EUR`. Request the same endpoint without the `currency` parameter to compare the pricing data against its `USD` equivalent.

Create an LCT Account

For LCT, you can leverage the traditional Accounts API to create any of the following account types:

- Fully Disclosed
- Omnibus
- Omnibus via the Alpaca Sub Ledger Solution

Below we provide an example of creating a account for a fully-disclosed setup with Euro as the local currency.

JSON


```
{
  "contact": {
    "email_address": "cool_alpaca_test@example.com",
    "phone_number": "555-666-7788",
    "street_address": ["20 N San Mateo Dr"],
    "city": "San Mateo",
    "state": "CA",
    "postal_code": "94401",
    "country": "USA"
  },
  "identity": {
    "given_name": "John",
    "family_name": "Doe",
    "date_of_birth": "1990-01-01",
    "tax_id": "666-55-4321",
    "tax_id_type": "USA_SSN",
    "country_of_citizenship": "USA",
    "country_of_birth": "USA",
    "country_of_tax_residence": "USA",
    "funding_source": ["employment_income"],
    "annual_income_min": "30000",
    "annual_income_max": "50000",
    "liquid_net_worth_min": "100000",
    "liquid_net_worth_max": "150000"
  },
  "disclosures": {
    "is_control_person": false,
    "is_affiliated_exchange_or_finra": false,
    "is_politically_exposed": false,
    "immediate_family_exposed": false
  },
  "agreements": [
   {
      "agreement": "customer_agreement",
      "signed_at": "2020-09-11T18:13:44Z",
      "ip_address": "185.13.21.99",
      "revision": "19.2022.02"
   },
   {
      "agreement": "crypto_agreement",
      "signed_at": "2020-09-11T18:13:44Z",
      "ip_address": "185.13.21.99",
      "revision": "04.2021.10"
    }
  ],
  "documents": [
    {
      "document_type": "identity_verification",
```

```
      "document_sub_type": "passport",
      "content": "/9j/Cg==",
      "mime_type": "image/jpeg"
    }
  ],
  "trusted_contact": {
    "given_name": "Jane",
    "family_name": "Doe",
    "email_address": "jane.doe@example.com"
  },
  "currency": "EUR"
}
```

Note the newly introduced `currency` parameter as part of the payload to create a new code.

Fund LCT Account

Accounts can be funded for LCT by either:

- Bank Wire
- Just in Time Cash
- Just In Time

The below example funds one of our Euro accounts created above, with JIT API `POST /v1/transfers/jit/transactions` with the following body:

JSON

```
{
  "account_id": "27529bc0-3ab5-34f5-ac29-54a98162472d",
  "entry_type": "JTD",
  "currency": "EUR",
  "amount": "500000",
  "description": "Test JIT EUR"
}
```

Calling the above mentioned API yields the following response,

JSON

```
{
    "id": "9a0ab8c2-4575-46b6-a6cc-f280c899b756",
    "account_id": "27529bc0-3ab5-34f5-ac29-54a98162472d",
    "created_at": "2022-08-31T16:29:44-04:00",
    "system_date": "2022-08-31",
    "entry_type": "JTD",
    "amount": "500000",
    "currency": "EUR",
    "description": "Test JIT EUR"
}
```

Estimate Stock Trade

Customers using LCT for the first time may not be sure how much their local currency can buy of a US stock. To address this pain point we created the Order Estimation Endpoint. The customer can enter:

- the security
- the notional value
- on the developer side you can input your swap rate to return the realistic value that your customer will receive.

We get in return indicative quantity, average price and USD value.

JSON


```
{
  "symbol": "AAPL",
  "side": "buy",
  "type": "market",
  "time_in_force": "day",
  "notional": "400",
  "swap_fee_bps": 45
}
```


The above payload will get an estimation for a market order to purchase AAPL stock with a notional amount of 400 EUR.

JSON


```
{
    "id": "b2f5f3f9-f6b9-4051-9e92-5872248f4830",
    "client_order_id": "b67e1ff9-794a-45b1-bd22-8d40f66c9f6a",
    "created_at": "2022-08-31T20:49:31.203137997Z",
    "updated_at": "2022-08-31T20:49:31.203137997Z",
    "submitted_at": "2022-08-31T20:49:31.200932908Z",
    "filled_at": "2022-08-31T20:49:31.200932908Z",
    "expired_at": null,
    "canceled_at": null,
    "failed_at": null,
    "replaced_at": null,
    "replaced_by": null,
    "replaces": null,
    "asset_id": "93f58d0b-6c53-432d-b8ce-2bad264dbd94",
    "symbol": "AAPL",
    "asset_class": "us_equity",
    "notional": "400",
    "qty": null,
    "filled_qty": "2.527276938",
    "filled_avg_price": "156.769502669363693758",
    "order_class": "",
    "order_type": "market",
    "type": "market",
    "side": "buy",
    "time_in_force": "day",
    "limit_price": null,
    "stop_price": null,
    "status": "filled",
```

```
    "extended_hours": false,
    "legs": null,
    "trail_percent": null,
    "trail_price": null,
    "hwm": null,
    "commission": "0",
    "swap_rate": "0.9948565977241001",
    "swap_fee_bps": "95",
    "subtag": null,
    "source": null,
    "usd": {
        "notional": "398.2483",
        "filled_avg_price": "157.58"
    }
}
```


Note the `usd` object at the bottom.

Submit a Stock Trade

Once having estimated a given order, we can actually commit to and execute the order using the usual Orders API.

We note here a few key LCT specific order attributes:

- Order `type` - must always be market
- `swap_fee_bps` - this is the correspondent spread. You as the correspondent can increase or decrease this as you require. **Note: Alpaca will have a separate spread**
- Quantity-based orders will also be accepted

JSON


```
{
  "symbol": "AAPL",
  "notional": "400",
  "side": "buy",
  "type": "market",
  "time_in_force": "day",
  "swap_fee_bps": 100
}
```


The response for the purchase of `AAPL` worth 400 EUR can be seen below,

JSON


```
{
    "id": "5e9ead9b-c73d-47d3-abcf-14301b4bc44c",
    "client_order_id": "6125fce1-ecf3-4ea9-a020-2f97f237fbca",
    "created_at": "2022-08-31T20:51:30.710557089Z",
    "updated_at": "2022-08-31T20:51:30.710618479Z",
    "submitted_at": "2022-08-31T20:51:30.70808831Z",
    "filled_at": null,
    "expired_at": null,
```

```
    "canceled_at": null,
    "failed_at": null,
    "replaced_at": null,
    "replaced_by": null,
    "replaces": null,
    "asset_id": "93f58d0b-6c53-432d-b8ce-2bad264dbd94",
    "symbol": "AAPL",
    "asset_class": "us_equity",
    "notional": "400",
    "qty": null,
    "filled_qty": "0",
    "filled_avg_price": null,
    "order_class": "",
    "order_type": "market",
    "type": "market",
    "side": "buy",
    "time_in_force": "day",
    "limit_price": null,
    "stop_price": null,
    "status": "pending_new",
    "extended_hours": false,
    "legs": null,
    "trail_percent": null,
    "trail_price": null,
    "hwm": null,
    "commission": "0",
    "swap_rate": "0.9947972168892468",
    "swap_fee_bps": "150",
    "subtag": null,
    "source": null,
    "usd": {
        "notional": "396.0606",
        "filled_avg_price": null
    }
}
```


Get Account Position

The below position is the `AAPL` stock purchased previosly with 400 EUR.

JSON


```
[
    {
        "asset_id": "93f58d0b-6c53-432d-b8ce-2bad264dbd94",
        "symbol": "AAPL",
        "exchange": "NASDAQ",
        "asset_class": "us_equity",
        "asset_marginable": false,
        "qty": "2.5132",
        "avg_entry_price": "156.7700934095764032",
        "side": "long",
        "market_value": "393.969232",
        "cost_basis": "393.9945987569474165523984",
        "unrealized_pl": "-0.0253667569474165523984",
        "unrealized_plpc": "-0.000064383514463",
        "unrealized_intraday_pl": "-0.0253667569474165223984",
        "unrealized_intraday_plpc": "-0.000064383514463",
        "current_price": "156.76",
```

```
        "lastday_price": "158.087157481573854921",
        "change_today": "-0.008395099910178",
        "swap_rate": "1.0048551416746259",
        "avg_entry_swap_rate": "0.9947972168892468",
        "usd": {
            "avg_entry_price": "157.59",
            "market_value": "392.0656974929108971",
            "cost_basis": "396.055188",
            "unrealized_pl": "-0.0252441928148389",
            "unrealized_plpc": "-0.000064383514463",
            "unrealized_intraday_pl": "-0.0252441928148389",
            "unrealized_intraday_plpc": "-0.000064383514463",
            "current_price": "156.0025853465346558",
            "lastday_price": "157.3233304236430846",
            "change_today": "-0.0083545374472457"
        },
        "qty_available": "2.5132"
    }
]
```


Journaling Local Currency

Journalling in LCT is almost exactly the same as our regular Journals API.

In this example we will journal some Euro between two accounts.

JSON


```
{
  "from_account": "51461a2a-8f98-3aa5-ae51-fad8d03037b3",
  "entry_type": "JNLC",
  "to_account": "27529bc0-3ab5-34f5-ac29-54a98162472d",
  "amount": "5000",
  "currency": "EUR",
  "description": "Test Euro Journal"
}
```


and the response

JSON


```
{
    "id": "1717b9c7-f516-4e85-a21b-bbeb7ef7a87a",
    "entry_type": "JNLC",
    "from_account": "51461a2a-8f98-3aa5-ae51-fad8d03037b3",
    "to_account": "27529bc0-3ab5-34f5-ac29-54a98162472d",
    "symbol": "",
    "qty": null,
    "price": "0",
    "status": "queued",
    "settle_date": null,
    "system_date": null,
    "net_amount": "5000",
    "description": "Test Euro Journal",
    "currency": "EUR"
}
```

SSE Events

Suggest Edits

Alpaca Broker API provides replayable and real-time event streams via Server-Sent Event (SSE). The SSE protocol is a simple yet powerful protocol to satisfy a lot of your needs to build flawless user experience. Each endpoint can be queried by the event timestamp or monotonically incremental integer ID to seamlessly subscribe from the past point-in-time event to the real-time pushes with a simple HTTP request. While all SSE endpoints follow the same JSON object model as other REST endpoints, SSE protocol is a lightweight addition on top of the basic HTTP protocol which is a bit different from REST protocol. Please make sure your client program handles the SSE protocol correctly.

Why Use SSE?

- Low Latency: Receive updates in real-time for timely decisions about your customers
- Resource Efficiency: A single connection serves multiple updates and streamlines where you receive updates about your customers
- Simplicity: Integration requires fewer lines of code compared to WebSockets.

Best Practices

- Connection Health: Implement heartbeat checks.
- Error Recovery: Code for auto-reconnection.
- Selective Listening: Subscribe to specific event types relevant to your use case.

Note about /v1 and /v2beta1

We are in the process of switching from integer IDs to ULIDs for our Events Streaming. ULIDs are designed to be lexicographically sortable, thanks to their structure that encodes a timestamp. This allows you to better sort and filter records based on when they occured. While they are more complex to read than integer IDs, they contain more information.

Currently only Admin Action Events and Trade Events leverage ULIDs, and over the next months we will be migrating the rest. Check back here to know which SSEs are on the new endpoint.

What Should You Do?

Legacy Events that still use an integer ID and have now an additional field called since_ulid and until_ulid. We highly recommend that you use those today so that you don't face any issues when we will eventually migrate the remaining events (account status, journal status, transfer status, trade status and non-trade-activity notifications) and deprecate the old ones.

Types of SSE Events

Account Status Events

Stay abreast of changes to account statuses. Learn more here.

You can find some sample responses below:

Equity & Crypto AccountEquity Only Account


```
{
    "account_blocked": false,
    "account_id": "9ab15e44-569c-4c32-952c-b83ab7076549",
    "account_number": "",
    "admin_configurations": {
        "allow_instant_ach": true,
        "disable_shorting": true
    },
    "at": "2023-10-13T13:34:28.30629Z",
    "crypto_status_from": "",
    "crypto_status_to": "APPROVED",
    "event_id": 12627517,
    "event_ulid": "01HCMKXQYJ3ZBV66Q21KCT1CRR",
    "pattern_day_trader": false,
    "status_from": "",
    "status_to": "APPROVED",
    "trading_blocked": false
}

{
    "account_id": "50333df9-66f0-46b9-a083-4212b152f749",
    "account_number": "307137914",
    "at": "2023-10-13T13:34:29.668043Z",
    "event_id": 12627518,
    "event_ulid": "01HCMKXS94ST351NFGEZR57EHV",
    "status_from": "APPROVED",
    "status_to": "ACTIVE"
}
```

```
:heartbeat

{
    "account_id": "9ab15e44-569c-4c32-952c-b83ab7076549",
    "account_number": "307645030",
    "at": "2023-10-13T13:35:18.145917Z",
    "crypto_status_from": "APPROVED",
    "crypto_status_to": "ACTIVE",
    "event_id": 12627519,
    "event_ulid": "01HCMKZ8M2XPNC9Y8HE159P2WK",
    "status_from": "APPROVED",
    "status_to": "APPROVED"
}

:heartbeat

{
    "account_id": "9ab15e44-569c-4c32-952c-b83ab7076549",
    "account_number": "307645030",
    "at": "2023-10-13T13:40:17.417798Z",
    "event_id": 12627521,
    "event_ulid": "01HCMM8CWAQETWNM75VJKA0YX2",
    "status_from": "APPROVED",
    "status_to": "ACTIVE"
}
```

Journal Events

Stay notified on the status of journal transactions to make sure they have been executed and the cash has been moved from one account to another. More details here.

You can find a sample response below:

JSON

```
{
    "at": "2023-10-13T13:11:10.57913Z",
    "entry_type": "JNLC",
    "event_id": 11751531,
    "event_ulid": "01HCMJK2ZKCPTYXMJYS66T0QJJ",
    "journal_id": "ddd26344-86af-4ba7-ae6a-bcec63129808",
    "status_from": "",
    "status_to": "queued"
}

{
    "at": "2023-10-13T13:11:10.634443Z",
    "entry_type": "JNLC",
    "event_id": 11751532,
    "event_ulid": "01HCMJK31AVBME4WNSH3C8E4HJ",
    "journal_id": "ddd26344-86af-4ba7-ae6a-bcec63129808",
    "status_from": "queued",
    "status_to": "sent_to_clearing"
}

{
```

```
    "at": "2023-10-13T13:11:10.67241Z",
    "entry_type": "JNLC",
    "event_id": 11751533,
    "event_ulid": "01HCMJK32GSBH2QG92TKZKDRRV",
    "journal_id": "ddd26344-86af-4ba7-ae6a-bcec63129808",
    "status_from": "sent_to_clearing",
    "status_to": "executed"
}
```


Transfer Events

Be notified instantly when the statuses of deposits and withdrawals are updated. Read further here.

You can find a sample response below:

JSON


```
{
  "account_id":"8e00606a-c9ac-409a-ba45-f55e8f77984a",
  "at":"2021-06-10T19:49:12.579109Z",
  "event_id":15960,
  "status_from":"",
  "status_to":"queued",
  "transfer_id":"c4ed4206-697b-4859-ab71-b9de6649859d"
}


{
  "account_id":"8e00606a-c9ac-409a-ba45-f55e8f77984a",
  "at":"2021-06-10T19:52:24.066998Z",
  "event_id":15961,
  "status_from":"queued",
  "status_to":"sent_to_clearing",
        "transfer_id":"c4ed4206-697b-4859-ab71-b9de6649859d"
}

{
  "account_id":"8e00606a-c9ac-409a-ba45-f55e8f77984a",
  "at":"2021-06-10T20:02:24.280178Z",
  "event_id":15962,
  "status_from":"sent_to_clearing",
  "status_to":"executed",
  "transfer_id":"c4ed4206-697b-4859-ab71-b9de6649859d"
}
```


Trade Events

Keep tabs on the status of orders, trades, and executions in real-time. Documentation here.

v2beta1v1


```
{
```

```
    "account_id": "aa4439c3-cf7d-4251-8689-a575a169d6d3",
    "at": "2023-10-13T13:28:58.387652Z",
    "event_id": "01HCMKKNRK7S5C1JYP50QGDECQ",
    "event": "new",
    "timestamp": "2023-10-13T13:28:58.37957033Z",
    "order": {
        "id": "bb2403bc-88ec-430b-b41c-f9ee80c8f0e1",
        "client_order_id": "508789e5-cea3-4235-b546-6c62ff92bd79",
        "created_at": "2023-10-13T13:28:58.361530031Z",
        "updated_at": "2023-10-13T13:28:58.386058029Z",
        "submitted_at": "2023-10-13T13:28:58.360070731Z",
        "filled_at": null,
        "expired_at": null,
        "cancel_requested_at": null,
        "canceled_at": null,
        "failed_at": null,
        "replaced_at": null,
        "replaced_by": null,
        "replaces": null,
        "asset_id": "b0b6dd9d-8b9b-48a9-ba46-b9d54906e415",
        "symbol": "AAPL",
        "asset_class": "us_equity",
        "notional": "10",
        "qty": null,
        "filled_qty": "0",
        "filled_avg_price": null,
        "order_class": "",
        "order_type": "market",
        "type": "market",
        "side": "buy",
        "time_in_force": "day",
        "limit_price": null,
        "stop_price": null,
        "status": "new",
        "extended_hours": false,
        "legs": null,
        "trail_percent": null,
        "trail_price": null,
        "hwm": null,
        "commission": "0"
    },
    "execution_id": "7922ab44-5b33-4049-ab9a-0cfd805ba989"
}

:heartbeat

{
    "account_id": "aa4439c3-cf7d-4251-8689-a575a169d6d3",
    "at": "2023-10-13T13:30:00.664778Z",
    "event_id": "01HCMKNJJRJ4E3RNFA1XR8CX7R",
    "event": "fill",
    "timestamp": "2023-10-13T13:30:00.658443088Z",
    "order": {
        "id": "db04069d-2e5a-48d4-a42f-6a0dea8ea0b8",
        "client_order_id": "be139e2d-8153-4ae8-83ee-7b98b4e17419",
        "created_at": "2023-10-13T13:22:21.887914Z",
        "updated_at": "2023-10-13T13:30:00.661902331Z",
        "submitted_at": "2023-10-13T13:23:05.411141Z",
        "filled_at": "2023-10-13T13:30:00.658443088Z",
        "expired_at": null,
```

```
        "cancel_requested_at": null,
        "canceled_at": null,
        "failed_at": null,
        "replaced_at": null,
        "replaced_by": null,
        "replaces": null,
        "asset_id": "b0b6dd9d-8b9b-48a9-ba46-b9d54906e415",
        "symbol": "AAPL",
        "asset_class": "us_equity",
        "notional": "10",
        "qty": null,
        "filled_qty": "0.05513895",
        "filled_avg_price": "181.36",
        "order_class": "",
        "order_type": "market",
        "type": "market",
        "side": "buy",
        "time_in_force": "day",
        "limit_price": null,
        "stop_price": null,
        "status": "filled",
        "extended_hours": false,
        "legs": null,
        "trail_percent": null,
        "trail_price": null,
        "hwm": null,
        "commission": "0"
    },
    "price": "181.36",
    "qty": "0.05513895",
    "position_qty": "0.05513895",
    "execution_id": "a958bb42-b034-4d17-bf07-805cf0820ffe"
}

{
    "account_id": "aa4439c3-cf7d-4251-8689-a575a169d6d3",
    "at": "2023-10-13T13:30:00.673857Z",
    "event_id": "01HCMKNJK1Y0R7VF6Q6CAC3SH7",
    "event": "fill",
    "timestamp": "2023-10-13T13:30:00.658388668Z",
    "order": {
        "id": "bb2403bc-88ec-430b-b41c-f9ee80c8f0e1",
        "client_order_id": "508789e5-cea3-4235-b546-6c62ff92bd79",
        "created_at": "2023-10-13T13:28:58.361530031Z",
        "updated_at": "2023-10-13T13:30:00.665807961Z",
        "submitted_at": "2023-10-13T13:28:58.360070731Z",
        "filled_at": "2023-10-13T13:30:00.658388668Z",
        "expired_at": null,
        "cancel_requested_at": null,
        "canceled_at": null,
        "failed_at": null,
        "replaced_at": null,
        "replaced_by": null,
        "replaces": null,
        "asset_id": "b0b6dd9d-8b9b-48a9-ba46-b9d54906e415",
        "symbol": "AAPL",
        "asset_class": "us_equity",
        "notional": "10",
        "qty": null,
        "filled_qty": "0.05513895",
```

```
        "filled_avg_price": "181.36",
        "order_class": "",
        "order_type": "market",
        "type": "market",
        "side": "buy",
        "time_in_force": "day",
        "limit_price": null,
        "stop_price": null,
        "status": "filled",
        "extended_hours": false,
        "legs": null,
        "trail_percent": null,
        "trail_price": null,
        "hwm": null,
        "commission": "0"
    },
    "price": "181.36",
    "qty": "0.05513895",
    "position_qty": "0.1102779",
    "execution_id": "33cbb614-bfc0-468b-b4d0-ccf08588ef77"
}
```


Message Ordering

For the messages received on the SSE stream we guarantee that the order of the received events is the same as the order they were happening on a per account basis.

Example: if event E1 has been received earlier then another event E2 for the same account, then E1 happened before E2 according to our bookkeeping.

We do not have this guarantee across accounts: if two events for different accounts are received it is the consumer's responsibility to decide which event happened first based on the timestamp/ulid fields of the event.

Example: E1 happened for account A1 before E2 which was affecting A2. The streaming endpoint might return the events in E1, E2 or E2, E1 ordering. Both responses should be considered valid.

Note: since ULIDs contain a random part other events might have arrived in the same millisecond as the last event received being lexiographicly less than the previous event.

If the stream is used for recon purposes, we recommend to restart the stream from a since that is a few mintues before the time of latest event received.

This approach means that the consumer will receive some events twice when restarting a stream: it is the consumer's responsibility to process the recevied

messages in an idempotent manner so that duplicate messages get ignored on the consumer side.

Note: since and until parameters are parsing as RFC3339 where timezone can be specified (e.g 2006-01-02T15:04:05+07:00), however plus sign character (+) is a special character in HTTP, so use the URL encoded version instead, e.g. ...events/trades?since=2006-01-02T15:04:05%2B07:00

Admin Action Events

These events pertain to administrative actions like account suspensions and liquidations performed by Alpaca Administrators. See more here.

Account status change


```
[
  {
    "event_id": "01GTVS4FVS2KJDTPYH2WM6NAXF",
    "at": "2023-09-21T10:52:38.429059991Z",
    "note": "Status changed to REJECTED.",
    "type": "legacy_note_admin_event",
    "context": {},
    "category": "other",
    "event_id": "03HBVNXKMWYGFTKTGNVR5R41F2",
    "correspondent": "ABCD",
    "belongs_to_kind": "account",
    "created_by_kind": "admin",
    "belongs_to_id_reference": "b4fe44b0-e51c-48f4-b674-990bea6cf8d7",
    "created_by_id_reference": "f0e150df-94ad-48f9-8b0f-05433a3b53c3"
  }
]
```


Non-Trade Activities Events

Covering non-trade activities like dividends, stock splits, and other corporate actions. Read more here.

You can find some sample responses below:

Cash DIVStock DIVDIVNRASPLITSPINCash MAStock MAREORGREG FEETAF FEECSD


```
{
  "id": "{GUID}",
  "qty": 0,
  "price": null,
  "status": "executed",
  "symbol": "BNDX",
  "entry_type": "DIV",
  "net_amount": 0.01,
  "description": "Cash DIV @ 0.0541, Pos QTY: 0.131126008, Rec Date: 2022-11-02",
  "settle_date": "2022-11-04",
```

```
  "system_date": "2022-11-04",
  "per_share_amount": 0.0541
}
```


Account Status Events for KYCaaS

Suggest Edits

For partners who utilize Alpaca's KYC service for opening brokerage accounts, if an account is moved to `ACTION_REQUIRED` or `APPROVAL_PENDING` then that indicates that additional action may be needed from you or your user to approve the account. These status updates, along with the reason for the status change, will be relayed in real time via the Account Status Events. The specific KYC results that may require action from your end user will wind up in `ACCEPT`, `INDETERMINATE`, or `REJECT`. The `additional_information` field will be used to relay custom messages from our account opening team. If a KYC result is returned via the `ACCEPT` object then no further action is needed to resolve the request. KYC results returned in the `INDETERMINATE` or `REJECT` objects will require further action before the account can be opened. The following tables can be used to determine what is required from the account opener.

Documentation Required

KYC Result Code	Government Issued ID Card	Tax ID Card	Statement (utility bill, etc.)	Sel
`IDENTITY_VERIFICATION`	**REQUIRED**			
`TAX_IDENTIFICATION`		**REQUIRED**		
`ADDRESS_VERIFICATION`	OPTIONAL		OPTIONAL	
`DATE_OF_BIRTH`	**REQUIRED**			
`SELFIE_VERIFICATION`				**RE**

Additional Information Required

KYC Result Code	Additional Information Required
`PEP`	Job title / occupation and address
`FAMILY_MEMBER_PEP`	Name of politically exposed person if immediate family
`CONTROL_PERSON`	Company name, company address, and company email
`AFFILIATED`	Company / firm name, company / firm address, company / firm email
`VISA_TYPE_OTHER`	Visa type and expiration date
`W8BEN_CORRECTION`	An updated W8BEN form with corrected information

KYC Result Code	Additional Information Required
`OTHER`	For specific cases our operational team might return with a customized message within additional_information attribute.

Broker API FAQs

Frequently Asked Questions

Suggest Edits

Broker API vs. Trading API

What is the Difference Between Trading API and Broker API?

The Broker API is meant to support different use cases for Broker-Dealers, RIAs, and Trading/Investing applications, see more here.

Instead, the Trading API is meant for Retail users and algotraders to communicate with Alpaca's brokerage service, see more here.

Is the Authentication Different Between Trading API and Broker API?

Yes, the way a client authenticates with the Broker API is different than the Trading API.

With the Broker API, you'll need to encode the keys and pass an authentication header that looks like this:

```
Authorization: Basic <base64 encoded keys>
```

see more here.

When using the Market Data API with Broker keys, you'll need to use the same authentication method that you would use with the Broker API.

Instead, with the Trading API, your authentication headers will use directly the keys this way:

```
"APCA-API-KEY-ID: {YOUR_API_KEY_ID}"
"APCA-API-SECRET-KEY: {YOUR_API_SECRET_KEY}"
```

see more here.

Accounts

When Do the Values on the Accounts Trading Get Updated?

All the values that **do not** have the prefix - “last_“ are updated Real-Time post trade executions and Corp Action activities. Might update outside market hours as well, post-trade settlements and FEE calculations like REG/TAF fee.

Assets

What is the difference between Tradable and Status Boolean in the Assets API?

Assets

There are some scenarios to consider here:

`tradable=true` and `status=active`

This is the scenario for most of the stocks listed. For example Apple Inc. This means it's tradable - both BUY and SELL are possible on this stock.

`tradable=false` and `status=active`

This is one of the rare conditions where we don't allow new opening transactions and only allow closing transactions. This usually happens when an asset is delisted from a US stock exchange and trades over the counter.

`tradable=false` and `status=inactive`

This is a scenario which can be achieved during the delisting of a company. If a stock is delisted, it is not tradable, which means we neither allow a `BUY` nor a `SELL` order for that particular stock. A delisted stock is either auto-liquidated or users are informed to close the positions within a specific amount of time.

Can OTC tickers be traded?

Yes, you can always close any open positions.

Are OTC symbols available in the Market Data?

No. OTC Market Data requires partners to directly have an agreement with OTC Markets. Once that is done, we can enable the same for our partners.

Is there a way to differentiate between Stocks and ETFs in the Assets API?

Alpaca at this point in time does not support any categorization of Stocks and ETFs. Partner needs to categorize them personally using the names and symbols.

Events

How Many SSE Connections Can be Alive Concurrently?

A maximum of **25** connection requests are allowed. Post that you will receive *"Too many requests"* error.

After How Long Will the SSE Connection Close?

There is no such time. Alpaca would never stop responding, hence we also send "*heartbeat*" to let partners know that the connection is alive. Sometimes the connections are un-knowingly closed by client-networks or network error happens. Nonetheless, you can always re-open and request data after a particular event using `since_id` or `since`.

What is the Use-case for SSE?

Server-sent events can be useful when you want updates on the status of a certain transaction, without making multiple HTTP requests. Events endpoints will return multiple responses when you open a connection, and can also be used in a restful way to make sure you didn't miss any event.

1. Account Status Updates - Subscribe to Account Status Events (SSE)
2. Trade Updates - Subscribe to Trade Events (SSE)
3. Journal Updates - Subscribe to Journal Events (SSE)
4. Any Transfer Event - Subscribe to Transfer Events (SSE)
5. Any relevant Non-Trading Activity - Subscribe to Non-Trading Activities Events (SSE)

Corporate Actions

Can I find sample events for possible corporate actions?

Absolutely yes. Sample events can be found here - Events | Alpaca Docs

Does the Market Data API support corporate actions adjustments?

Yes, price adjustments are visible on the Market Data, you'll just need to make sure the adjustment parameter is set to all. The available query parameters for bars are available here - Historical bars

Does the Market Data API support local currencies?

Yes. You can retrieve historical market data in the supported local currencies.

Symbol Changes

Are symbol changes notified via the SSE Events?

No, as of now the symbol changes need to be seen via the Assets API. Syncing at regular intervals is what is recommended to make sure you are up-to-date with the symbol changes.

Do we have a new asset object, if there is a symbol change?

The answer is a YES and a NO. The reason for that is, if there is simply a symbol change without the CUSIP being updated, in that case, we just update the existing asset by updating the Symbol. If there is a CUSIP change, the current asset becomes INACTIVE and a new Asset Object is added the the response of the Assets API.

Documents

How are trade confirmations and account statements handled by Alpaca and its partners?

1. Both of them need to be mailed to users with auto-archive-bd@alpaca.markets in BCC.
2. The email should contain the truncated alpaca account number and language as mentioned in the Statement and Confirms Document.
3. If sharing a customized statement or confirmation, in that case, the partner must allow users to have access to the original statements and confirms generated by Alpaca.

When are these documents available?

Daily Trade Confirmations are available on the next day after the BOD job is finished(02:15 AM-02:30 AM EST).

Monthly statements are available after the 1st weekend of the next month.

How can I retrieve these documents?

Documents | Alpaca Docs

Query the above-mentioned endpoint
with `type` as `account_statement` or `trade_confirmation`.

Can I find a sample W-8 Ben Form and what details are needed to be filled in?

Can be found here - https://github.com/alpacahq/bkdocs/files/7756721/W-8Ben_Example_Broker_Docs.pdf

I'm getting a 400 error (request body format is invalid: JSON: cannot unmarshal object into Go value of type entities.BrokerAccountDocumentUploads) when uploading a W8-Ben. Is this expected?

Yes, the body of the request is an array of documents, make sure you're using square brackets like in the sample response Documents | Alpaca Docs.

Journals

How much time does JNLC's or Journal usually take?

Most of the time, JNLCs are pretty fast and mostly instantaneous, with a sub-second average delay before the execution is complete.

There can be a delay of a few hours when exceeding the limits mentioned below.

JNLC with larger amounts are executed the next day. Is this expected?

Yes. We have a limit called the JNLC Transaction Limit. This is the maximum amount of journaling you can do instantaneously. Any amount of journal greater than this would be executed as a part of the BOD job of the next day. The default value is $50.

This is a safeguard mechanism to prevent LARGE withdrawals for any user to prevent partners from the same. This is a “Configurable“ value that can be set to anything on the Sandbox but would require a discussion for the PROD.

There is also a JNLC Daily Limit, which is the summation of all the JNLC done in a day. If the limit is $1000(Default Limit), you can do 10 transactions of $100 each, but the 11th one would not pass. This is a “Configurable“ value that can be set to anything on the Sandbox but would require a discussion for the PROD.

Can I subscribe to a journal status streaming without having to check it again and again?

Absolutely yes. You can listen to the journal server-sent events. - Events | Alpaca Docs

What are the use cases for the JNLS?

JNLS are usually used to Reward users for signing up on the platform by our partners.

What is the direction and flow of JNLS?

JNLS can ONLY be done from the SWEEP ACCOUNT to the USER ACCOUNT and not vice-versa. Once, done, it can’t be sent back.

Is the JNLS transactional limit per account configurable?

Yes, they are configurable.

Are JNLS transactional limits per account USD or Stocks based?

They are USD-based.

Trading

Orders

What is the minimum order value?

All buy orders must have a minimum market value of 1$ or the request will be rejected with a 422. Sell orders do not have this validation, so you can close any open position.

What is the precision for notional and qty in the order request body?

A maximum of **2 decimal places** for *notional* and **9 decimal places** for *qty* should be used to send orders.

Only limit is allowed as the order type for OCO. Does this rule apply to OTO?

No. OCO implements an order cancellation and hence it might be very difficult to cancel a market order. Whereas in the case of OTO, that isn't an issue and hence *market* and *limit* both orders are allowed!

Order replacement is supported for OCO to update limit_price and stop_price.

Can we update the quantity as well? And can we use a notional amount instead of quantity?

Yes, you can update the quantity while patching the order using this API - Orders | Alpaca Docs .

No, you can't use the notional amount. The reason for this is, that Alpaca at this point of time **doesn't** support *LIMIT* orders for *notional* orders and since OCO orders can only be a *LIMIT* order, it makes it compulsory to use the **quantity**.

What orders can be cancelled?

We check that the order is in an "open" status before attempting to cancel it. The statuses that we consider open are:

- Accepted
- New
- Partially Filled
- Calculated
- Pending New
- Pending Cancel
- Pending Replace
- Accepted for Bidding

If the order is in one of these statuses you should receive a 204 (unless there was a server error) while if it isn't you're going to receive a 422, as mentioned in the docs.
To check if the order was actually canceled, I suggest you check its status on the trades SSE or through the REST endpoints.

Trade Settlement

Why is there no cash_withdrawable and cash_transferable increase after selling a stock?

Settlement occurs two business days after the day the order executes, or T+2 (trade date plus two days). For example, if you were to execute an order on Monday, it would typically settle on Wednesday. The `cash` is updated post the SELL trade is filled, but the `cash_withdrawable` and `cash_transferable` are updated post T+2.

What are some fees, that can be expected while trading stocks?

REG and TAF Fees

This is charged(REG and TAF individually) to a User Account, every time a SELL Order is executed. This is charged by Alpaca but, is a pass thru fee from the FINRA and SEC itself that is levied to Alpaca on sell orders. The REG and TAF fees change rates periodically. More about this can be read in the blog from our co-founder - REG/TAF Fee Updates with Alpaca

ADR Fees

We typically charge ADR fees once per month as Velox usually debits our account once per month. Most ADRs charge a fee of .01 to .03 per share once or twice a year(but it can be different as well). The record date on the charge is usually some time after the actual date of the charge. This can lead to issues where a client is charged for an ADR when they had sold it weeks prior.

Rounding rules

We will not charge the ADR fee if the calculated fee is less than a penny. However, if the amount is over a penny we will always round up to the next highest penny. For example, if the charge is .02 cents a share and the client owns 10.1 shares we would charge the client .21 for the ADR fee.

Rebalancing

Can I enter manual trades while using the Rebalancing API?

If the account is subscribed to a portfolio, it cannot execute manual trades or manual rebalance runs. Still, you can delete the subscription to execute manual trades or rebalance runs.

How can I see the orders triggered by a rebalancing event?

You can easily retrieve all the runs with this endpoint or a particular run by its ID, where each run will have an array with all the orders that were executed and the ones that failed or were skipped.

Important Links and Reads

1. Alpaca Daily Processes and Reconciliations - Daily Processes and Reconcilations
2. PDT - User Protection
3. Margins - Margin and Short Selling
4. Understanding Orders and Time in Force - Orders at Alpaca

Example Trading App (Ribbit)

Suggest Edits

What is Ribbit?

Ribbit is a skeleton mobile app designed to showcase the capabilities of the Broker API. It's a fully functional trading application that demonstrates how users would interact with your product. It uses all the different functionality that

the Broker API offers including onboarding new users, funding an account, managing market data, and handling trade activity.

User Experience Example

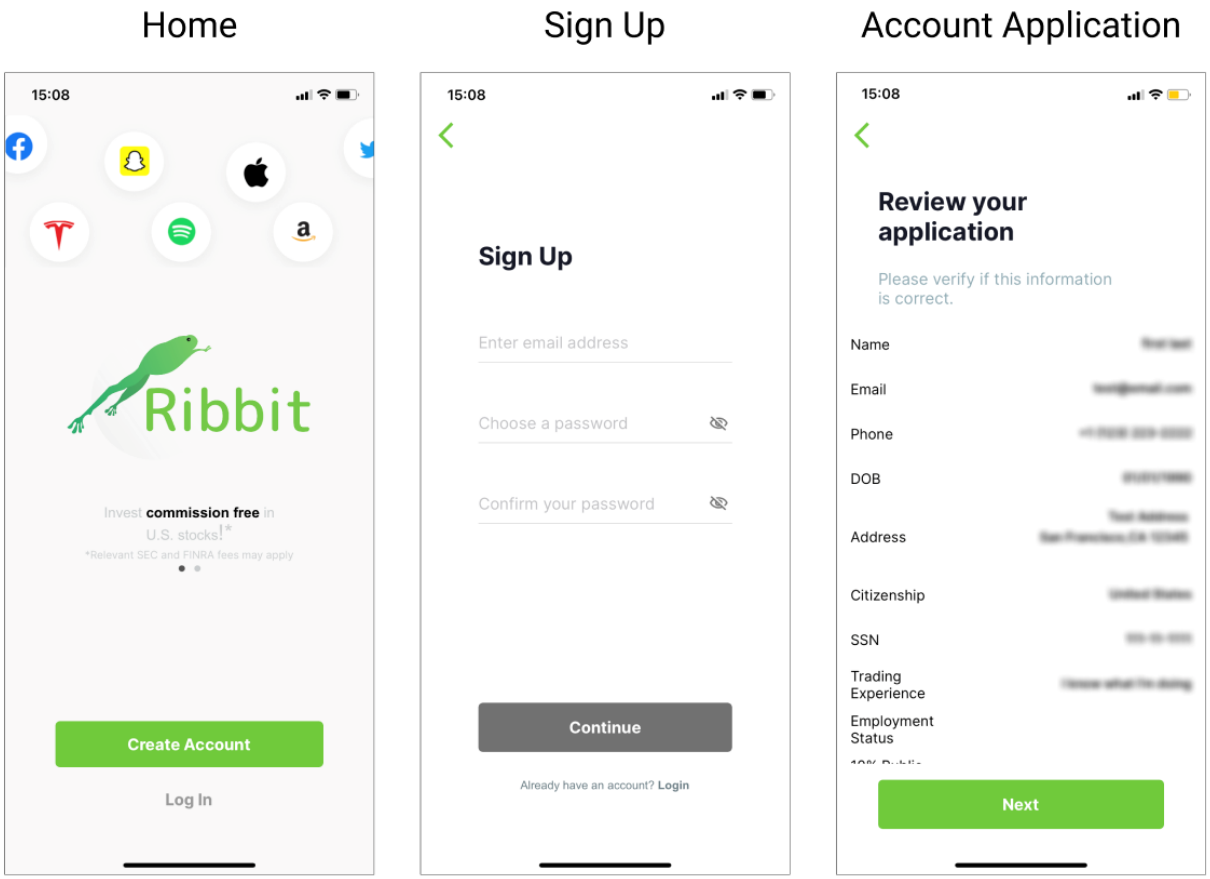
The screenshots below demonstrate how a native user would walk through Ribbit to accomplish various tasks.

Create a New Account

Once Ribbit users sign up with their email and create a password, it triggers the brokerage account onboarding process to begin. The following screens prompt users to input their information such as name, date of birth, tax ID, and more information that is required by law to open a brokerage account. At the end of this process, Ribbit calls the Accounts API to submit all the information to Alpaca where we verify the information and approve the account application.

The app demonstrates a common flow that brokerage apps have to implement to collect all the necessary data points and required user agreements. For your own app, you may also be interested in performing various input checks on the client side so that the account approval process is as quick as possible. See below screenshots of the actual flow.

Once the account creation flow is completed, Ribbit continues to use the Accounts API to retrieve real-time information about the user's account. The API can also be used to update the account information as well as request to close an account.



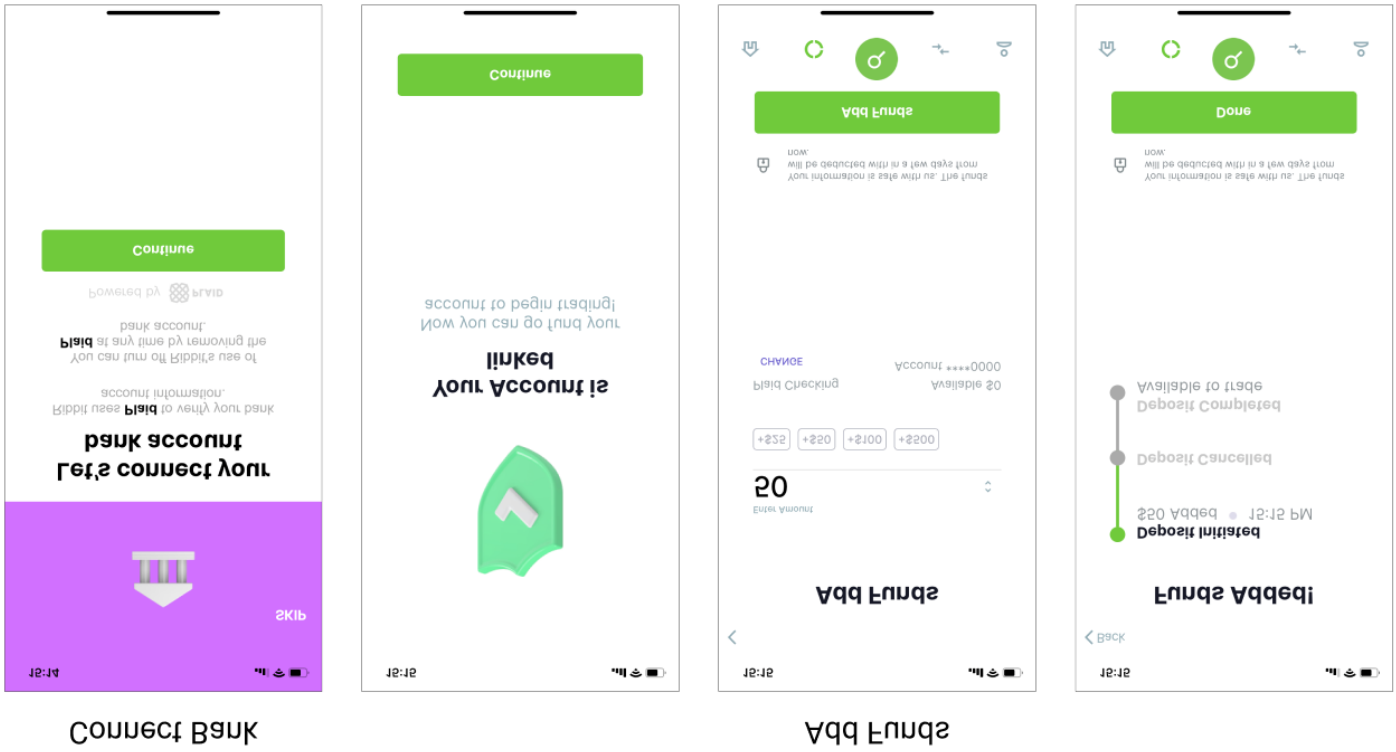
Beginning and end state of an account opening experience

Fund an Account

The next step for the new users is to deposit the money to start trading. Ribbit uses Plaid to validate the bank information so that Alpaca can simply link the bank account to the brokerage account. From the Plaid Link component, Ribbit receives the bank routing number and account number for the user and submits the bank link request using ACH Relationships.

As a demo app, Ribbit uses the Plaid sandbox which simulates the production environment behavior. When you try the app, use `user_good` and `pass_good` for the credentials with any banks shown in the app. Alpaca's sandbox where Ribbit simulates the ACH transactions and the virtual money is credited in the user's account in a moment.
Allowing your end users to connect to their personal bank and fund their account on your app can be intimidating if you aren't familiar with the high level financial requirements and flows. Fortunately, our Bank, ACH Relationships, and Transfers APIs make it easy to achieve this! The Bank API lets you create, retrieve, and delete bank relationships between their personal bank and their account on your

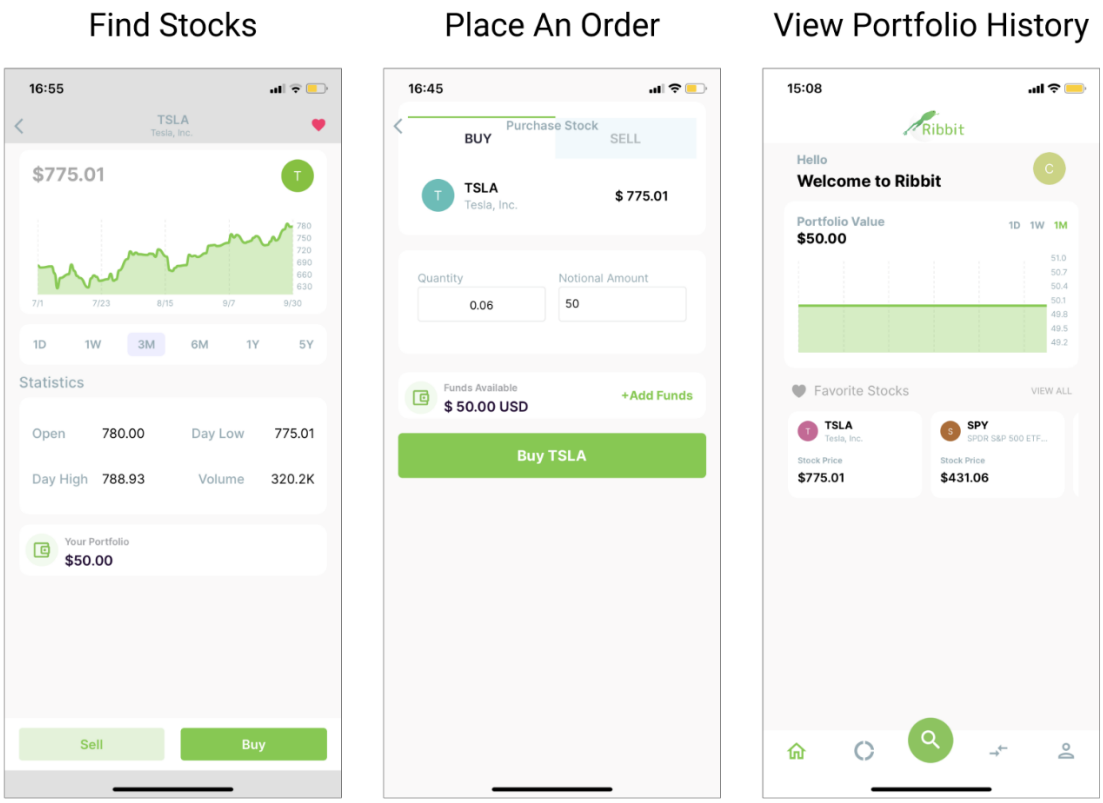
app. The ACH Relationships API deals with connecting, getting, and deleting your end user's specific bank account that will be used to initiate and receive ACH transfers from your app. Finally, the Transfers API initiates, lists, and cancels the actual transfer initiated from your app on behalf of your end user. See how this flow is implemented from your user's perspective below.



Example of a funding flow using Plaid

View and Execute Trades

When it comes to managing stock market data, Alpaca provides seamless integration via the Market Data API. Ribbit uses the historical data endpoint to draw the chart in the individual stock screen, and the real-time data endpoint to show the most up-to-date price information in the order screen. See how Ribbit makes use of the Market Data API below.



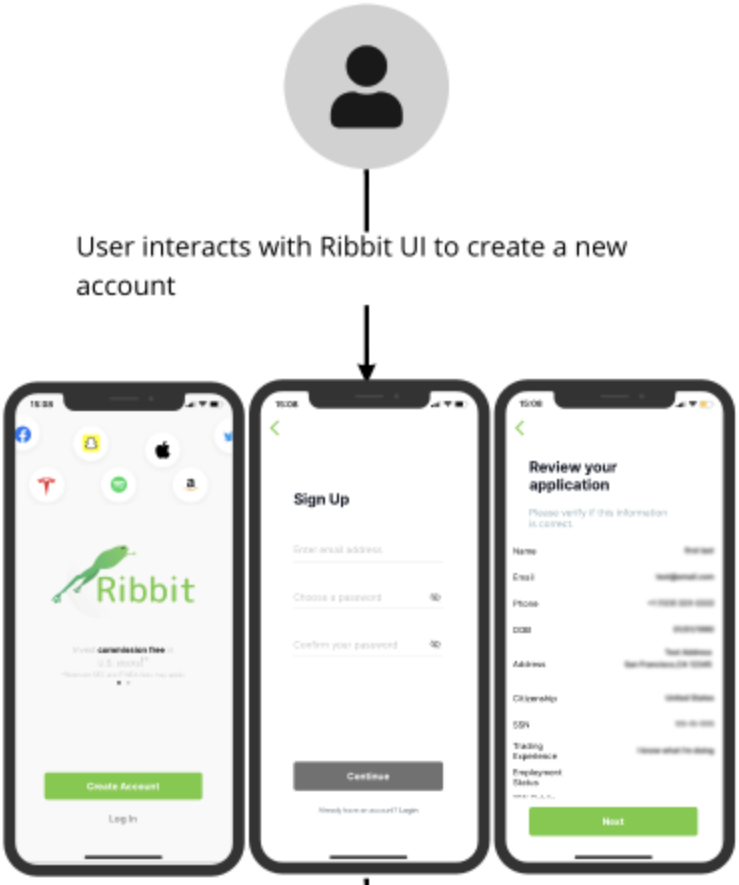
Typical screens for trading and portfolio

In the order screen, Ribbit uses the Orders API. It allows you to submit a new order, replace/cancel an open order, and retrieve a list of orders from a user's history. Ribbit connects to Alpaca's sandbox environment where an order execution simulator engine runs. This simulator will take the order you submitted on the backend and execute it using the real-time market price which makes it easy to test trading functionality before you launch your app to users.

Ribbit shows all the account activities using the Activities API which returns the relevant transaction history for a given account. As a trading app, some of the important requirements to deliver to your users are monthly statements and trade confirmations. Ribbit accomplishes this by using the Documents API. The documents are generated in PDF format by Alpaca so all you need to do is call the API to retrieve the list of downloadable URLs and show them in the app.

Architecture

The end user interacts with Ribbit's UI to achieve a task while Ribbit's backend processes the requests by making calls to Broker API. See the diagram below for an example of how the account creation process works.



User interacts with Ribbit UI to create a new account

Ribbit backend calls account creation request

POST /v1/accounts

Ribbit backend receives confirmation that account was created


```
{
  "account": {
    "id": "0d18ae51-3c94-4511-b209-101e1666416b",
    "account_number": "9034005019",
    "status": "ACTIVE",
    "currency": "USD",
    "last_equity": "1500.65",
    "created_at": "2019-09-30T23:55:31.185998Z"
  }
}
```

The backend application serves as a thin layer to proxy the API requests coming from the mobile app but makes sure each request is authorized for the appropriate user.

Technology

The user interface is written in Swift for iOS and Java for Android. The backend is implemented using Go.

Alpaca APIs

All of the technology that is needed for users to interact with Ribbit's core functionality is acheived through the Broker API. Accessing information related to the market is gathered using the Market Data API.

Where Can I Access the Source Code?

The codebase is hosted on GitHub and separated into three different repositories for the implementation of the backend, iOS user interface, and Android user interface

Options Trading Overview

Suggest Edits

Initial Options Offering

Supported

- US listed equities options, all american style
- Level 1+2 options trading
- Fully Disclosed partner relationships
- Automatic account approval process via the API along with SSE events
- Ability for partners to downgrade/disable options trading
- Ability to exercise options via the API
- DNE (do not exercise) requests by email

- New options specific activities
- Access to options market data (at a cost) or referral to market data partner

Not supported

- Omnibus partner relationships (coming soon)
- Level 3 + options strategies (multi-leg, spreads, naked, etc)
- LCT (local currency trading)
- Fractional options
- Extended hours

Options Enablement

Enabling Options Trading on an Account

Per FINRA Rule 2360, each customer account has to be approved for options before the first options trade is made. Once options trading is enabled on a correspondent you can start requesting accounts to be approved for options trading. For existing accounts there will be a few new data points that need to be patched on the account along with a new options agreement that the customer will need to sign. Below are the new fields that will need to be provided specifically for options.

- Annual Income
- Net Worth
- Liquid Net Worth
- Liquidity Needs
- Investment Experience
- Investment Risk Tolerance
- Investment Objective
- Investment Time Horizon

Below is an example of a request for opening a new account with options enabled. More details on creating an account can be found here.

JSON

```
{
```

```
"enabled_assets": [
    "us_equity",
    "crypto",
    "us_option"
],
"contact": {
    "email_address": "joe+smith+options+19@alpaca.markets",
    "phone_number": "555-666-7788",
    "street_address": [
        "20 N San Mateo Dr"
    ],
    "unit": "Apt 1A",
    "city": "San Mateo",
    "state": "CA",
    "postal_code": "94401",
    "country": "USA"
},
"identity": {
    "given_name": "John",
    "middle_name": "Smith",
    "family_name": "Doe",
    "date_of_birth": "1990-01-01",
    "tax_id": "124-55-4321",
    "tax_id_type": "USA_SSN",
    "country_of_citizenship": "USA",
    "country_of_birth": "USA",
    "country_of_tax_residence": "USA",
    "funding_source": [
        "employment_income"
    ],

    "annual_income_min": "10000",
    "annual_income_max": "10000",
    "total_net_worth_min": "10000",
    "total_net_worth_max": "10000",
    "liquid_net_worth_min": "10000",
    "liquid_net_worth_max": "10000",
    "liquidity_needs": "does_not_matter",
    "investment_experience_with_stocks": "over_5_years",
    "investment_experience_with_options": "over_5_years",
    "risk_tolerance": "moderate",
    "investment_objective": "market_speculation",
    "investment_time_horizon": "more_than_10_years"
},
"disclosures": {
    "is_control_person": false,
    "is_affiliated_exchange_or_finra": true,
    "is_politically_exposed": false,
    "immediate_family_exposed": false,
    "context": [
        {
            "context_type": "AFFILIATE_FIRM",
            "company_name": "Finra",
            "company_street_address": [
                "1735 K Street, NW"
            ],
            "company_city": "Washington",
            "company_state": "DC",
            "company_country": "USA",
            "company_compliance_email": "compliance@finra.org"
```

```
            }
        ]
    },
    "agreements": [
        {
            "agreement": "margin_agreement",
            "signed_at": "2020-09-11T18:09:33Z",
            "ip_address": "185.13.21.99",
            "revision": "16.2021.05"
        },
        {
            "agreement": "account_agreement",
            "signed_at": "2020-09-11T18:13:44Z",
            "ip_address": "185.13.21.99",
            "revision": "16.2021.05"
        },
        {
            "agreement": "customer_agreement",
            "signed_at": "2020-09-11T18:13:44Z",
            "ip_address": "185.13.21.99",
            "revision": "16.2021.05"
        },
        {
            "agreement": "crypto_agreement",
            "signed_at": "2020-09-11T18:13:44Z",
            "ip_address": "185.13.21.99",
            "revision": "04.2021.10"
        },
        {
            "agreement": "options_agreement",
            "signed_at": "2020-09-11T18:13:44Z",
            "ip_address": "185.13.21.99"
        }
    ],
    "documents": [
        {
            "document_type": "identity_verification",
            "document_sub_type": "passport",
            "content": "/9j/Cg==",
            "mime_type": "image/jpeg"
        }
    ],
    "trusted_contact": {
        "given_name": "Jane",
        "family_name": "Doe",
        "email_address": "jane.doe@example.com"
    }
}
```


Options Approval Process

Once an account request is created we will need to manually approve the account for level 1 or 2 but in the future it will be auto approved based on the answers provided to the options specific onboarding fields.

Downgrading/disabling Options

Once an account is approved for options trading the customer has the ability to downgrade the options level or disable it altogether. This is done by setting the max_options_trading_level via the trading account configuration endpoint.

Once the max options trading level is set the options trading level on the trading account endpoint will get upgraded to the value set by the user. Note the options trading level will always be the same or lower than the options approved level. Below is an example of a user being approved for level 2 but downgrading to level 1.

JSON

```
{
  ...
  "options_approved_level": 2,
  "options_trading_level": 1,
  ...
}
```

Trading

Trading Overview

When placing orders via the orders api for options below is what is supported and what is not supported

Supported

- Options symbol
- Time in force of day
- Market, limit, stop, stop limit order types
- Ability to replace and cancel orders
- Level 1 + 2 option strategies

Not supported

- Extended hours
- Fractional or notional order support

Trading Levels

Alpaca supports the below options trading levels.

Level	Supported Trades	Validation
0	- Options trading is disabled	- NA
1	- Sell a covered call - Sell cash-secured put	- User must own sufficient underlying shares - User must have sufficient options buying power
2	- Level 1 - Buy a call - Buy a put	- User must have sufficient options buying power

Trading Level Validation

If a user tries to trade a strategy above their level this will result in an error message. Below is an example of a user who is approved for level 1 trying to place a level 2 options trade.

JSON


```
{
   "code": 40310000,
   "message": "account not eligible for level2 options trading"
}
```


Asset Master

Similar to the asset master for securities and crypto there is an options contract endpoint which will return all the options contract for an underlying symbol. Below is a sample

JSON


```
{
  "id": "1fb904df-961a-4a07-a924-53a437626db2",
  "symbol": "AAPL240223C00095000",
  "name": "AAPL Feb 23 2024 95 Call",
  "status": "active",
  "tradable": true,
  "expiration_date": "2024-02-23",
  "root_symbol": "AAPL",
  "underlying_symbol": "AAPL",
  "underlying_asset_id": "b0b6dd9d-8b9b-48a9-ba46-b9d54906e415",
  "type": "call",
  "style": "american",
  "strike_price": "95",
  "size": "100",
  "open_interest": "12",
  "open_interest_date": "2024-02-22",
  "close_price": "89.35",
  "close_price_date": "2024-02-22"
```

```
}
```

Order Examples

Buying a call

JSON

```
{
  "symbol": "PTON240126C00000500",
  "qty": "1",
  "side":"buy",
  "type": "market",
  "time_in_force": "day"
}
```

Buying a put

JSON

```
{
  "symbol": "TSLA240126P00210000",
  "qty": "1",
  "side": "buy",
  "type": "market",
  "time_in_force": "day"
}
```

Selling a covered call

JSON

```
{
  "symbol": "AAPL240126C00050000",
  "qty": "1",
  "side": "sell",
  "type": "market",
  "time_in_force": "day"
}
```

Selling a cash secured put

JSON

```
{
  "symbol": "QS240126P00006500",
  "qty": "1",
  "side": "sell",
  "type": "market",
  "time_in_force": "day"
}
```

Buying Power Checks

With options we introduce a new field on the trading account endpoint called options_buying_power which is evaluated when trying to open new options positions.

For both **buying a call** or **buying a put** sufficient buying power is necessary to pay for the premium of the contract otherwise an error will occur as shown below.

JSON


```
{
"options_buying_power": "7267.84",
"code": 40310000,
"cost_basis": "38000",
"message": "insufficient options buying power"
}
```


For **Selling Covered Call** you need to have enough stocks as collateral otherwise the order will error out as shown below.

JSON


```
 {
   "available_qty": "0",
   "code": 40310000,
   "message": "insufficient underlying qty available for covered call (required:
100, available: 0)",
   "required_qty": "100",
   "symbol": "AAPL240126C00050000",
   "underlying_asset_id": "b0b6dd9d-8b9b-48a9-ba46-b9d54906e415"
}
```


For **selling cash secured put** you will need enough buying power to pay for the underlying stock minus the premium that would be received otherwise the order will error out as shown below

JSON


```
{
  "buying_power": "9395",
  "code": 40310000,
  "message": "insufficient options buying power for cash-secured put (required:
20310, available: 9395)",
  "required_options_buying_power": "20310"
}
```


Positions

Option positions will show up like any other position in the positions endpoint. Below is an example of an options position.

JSON


```
{
```

```
  "asset_id": "fe4f43e5-60a4-4269-ba4c-3d304444d58b",
  "symbol": "PTON240126C00000500",
  "exchange": "",
  "asset_class": "us_option",
  "asset_marginable": true,
  "qty": "2",
  "avg_entry_price": "6.05",
  "side": "long",
  "market_value": "1068",
  "cost_basis": "1210",
  "unrealized_pl": "-142",
  "unrealized_plpc": "-0.1173553719008264",
  "unrealized_intraday_pl": "-142",
  "unrealized_intraday_plpc": "-0.1173553719008264",
  "current_price": "5.34",
  "lastday_price": "5.34",
  "change_today": "0",
  "qty_available": "2"
}
```


Post Trade

Exercising

As the buyer of an option (call or put), the holder has the right, but not the obligation, to buy or sell the option's underlying security at a specified price on or before a specified date in the future. If the holder decides to act on those rights they are choosing to exercise. An option can be exercised via the exercise endpoint and Alpaca will accept DNE(do not exercise) instructions via email to avoid exercising at expiration. Note exercise requests will be processed instantly and requests sent outside of market hours will be rejected. When exercising there will be 2 new activities per exercise which are:

- **OPEXC** - removes the options position as a result of exercising
- **OPTRD** - trading activity that is paired with the exercising

Exercising Call

A long call exercise results in **buying the underlying stock** at the strike price. Below is an example of this.

JSON


```
{
  "id": "20240227000000000::197118f0-afd8-4adb-b154-167f4a87b1f5",
  "account_id": "21c36b57-2304-4e5b-8364-8effbcba853e",
  "activity_type": "OPEXC",
  "date": "2024-02-27",
  "net_amount": "0",
```

```
  "description": "Option Exercise",
  "symbol": "QS240301C00006500",
  "qty": "-1",
  "status": "executed"
},

{
  "id": "20240227000000000::aa97cbc4-5163-49ab-b832-682a3a3e85bb",
  "account_id": "21c36b57-2304-4e5b-8364-8effbcba853e",
  "activity_type": "OPTRD",
  "date": "2024-02-27",
  "net_amount": "-650",
  "description": "Option Trade",
  "symbol": "QS",
  "qty": "100",
  "price": "6.5",
  "status": "executed"
}
```


Exercising Put

A long put exercise results in **selling the underlying stock** at the strike price. Below is an example of this.

JSON


```
{
  "id": "20240227000000000::f62ee8f5-0279-4e81-9bd1-4ef197c7b2f3",
  "account_id": "21c36b57-2304-4e5b-8364-8effbcba853e",
  "activity_type": "OPEXC",
  "date": "2024-02-27",
  "net_amount": "0",
  "description": "Option Exercise",
  "symbol": "QS240301P00006500",
  "qty": "-1",
  "status": "executed"
},

{
  "id": "20240227000000000::74e1db8e-9316-4dcf-b69c-50f51427b7c1",
  "account_id": "21c36b57-2304-4e5b-8364-8effbcba853e",
  "activity_type": "OPTRD",
  "date": "2024-02-27",
  "net_amount": "650",
  "description": "Option Trade",
  "symbol": "QS",
  "qty": "-100",
  "price": "6.5",
  "status": "executed"
}
```


Assignment

When selling an options contract (call or put) the user collects a premium but in turn takes on the obligation to buy or sell the stock at the agreed upon strike

price if assigned. It is important to note that the OCC assigns a random account and sellers of contracts can be assigned overnight. When assigned there will be 2 new activities per assignment which are:

- **OPASN** - removes the options position as a result of assignment
- **OPTRD** - trading activity that is paired with the assignment

Call Assignment

For call options, the seller takes on the **obligation to sell** the stock at the agreed upon strike price. Below is an example of activities that result as a result of a call assignment

JSON


```
 {
   "id": "20240301000000000::001140db-3947-456b-aefc-253861fb65df",
   "account_id": "21c36b57-2304-4e5b-8364-8effbcba853e",
   "activity_type": "OPASN",
   "date": "2024-03-01",
   "net_amount": "0",
   "description": "",
   "symbol": "QS240301C00004500",
   "qty": "1",
   "status": "executed"
}

{
  "id": "20240301000000000::a88c089f-a8c3-4672-9b68-2f6f2d05e914",
  "account_id": "21c36b57-2304-4e5b-8364-8effbcba853e",
  "activity_type": "OPTRD",
  "date": "2024-03-01",
  "net_amount": "450",
  "description": "Option Trade",
  "symbol": "QS",
  "qty": "-100",
  "price": "4.5",
  "status": "executed"
}
```


Put Assignment

For put options, the seller takes on the **obligation to buy** the stock at the agreed upon strike price. Below is an example of activities that result as a result of a put assignment

JSON


```
{
  "id": "20240301000000000::fcd92e5c-46e4-4c6e-8866-d56cd7d2bde2",
  "account_id": "21c36b57-2304-4e5b-8364-8effbcba853e",
  "activity_type": "OPASN",
  "date": "2024-03-01",
```

```
  "net_amount": "0",
  "description": "",
  "symbol": "QS240301P00009000",
  "qty": "1",
  "status": "executed"
}
```

```
{
  "id": "20240301000000000::1e1c2804-ce68-4516-9fa4-62d49b14c334",
  "account_id": "21c36b57-2304-4e5b-8364-8effbcba853e",
  "activity_type": "OPTRD",
  "date": "2024-03-01",
  "net_amount": "-900",
  "description": "Option Trade",
  "symbol": "QS",
  "qty": "100",
  "price": "9",
  "status": "executed"
}
```


Expiration

Unlike stocks, option contracts have an expiration date. If the user does not close out their position or exercise their options (if they have the buying power to support the exercise), Alpaca will need to potentially take action on it for risk management purposes based on the criteria described below.

- Starting at 3:00 pm EST on the date of expiration, Alpaca continuously evaluates any open positions that expire that day and stops accepting new orders to open/extend any positions.
- If the position for a long call or put **is ITM (in the money)** by .01 or more** **, Alpaca checks if the account has enough buying power to exercise
    - If there is **enough buying power** Alpaca will not liquidate the options and will let the option auto-exercise.
    - If there is **not enough buying power**, then Alpaca will liquidate while it's still ITM
- If the position **is for a covered call or cash secured put and ITM** by .01 or more** **Alpaca will automatically assign after close on date of expiry** **
- If the position **is not ITM** in other words it's **ATM (**at the money) or **OTM** (out of the money), Alpaca may skip it and it will expire after close as shown in the examples below. (Note that positions slightly OTM may also be liquidated depending on market and underlying conditions. Alpaca will take into consideration fast moving markets

and/or fast stocks that may be moving from ITM and OTM throughout the day. There may be instances where OTM positions are also closed out.)

Call Expiration

JSON


```
{
  "id": "20240301000000000::31ff5d4c-a608-43bb-8e59-678a96a4a42c",
  "account_id": "21c36b57-2304-4e5b-8364-8effbcba853e",
  "activity_type": "OPEXP",
  "date": "2024-03-01",
  "net_amount": "0",
  "description": "Option Expiry",
  "symbol": "QS240301C00010500",
  "qty": "-1",
  "status": "executed"
}
```


Put Expiration

JSON


```
{
  "id": "20240301000000000::4bd8c683-0b2c-489f-9df9-1243ff621648",
  "account_id": "21c36b57-2304-4e5b-8364-8effbcba853e",
  "activity_type": "OPEXP",
  "date": "2024-03-01",
  "net_amount": "0",
  "description": "Option Expiry",
  "symbol": "QS240301P00000500",
  "qty": "-1",
  "status": "executed"
}
```


Market Data

Alpaca provides access to options market data (at a cost) in the same user-friendly format it does for stocks and crypto. Additionally, we can refer partners to external data vendors with whom Alpaca has a close relationship if necessary.

Endpoint	Noteste
Latest Quotes	Latest quotes. Supports multiple symbols
Latest Trades	Latest trades. Supports multiple symbols
Historical Trades	Historical trades. Supports multiple symbols
Snapshots	Combine quotes and trades for multiple symbols

Endpoint	Noteste
Option chain	Query option chain by the underlying (stock) symbols
Historical bars	Historical bars.Supports multiple symbol

Options trading is not suitable for all investors due to its inherent high risk, which can potentially result in significant losses. Please read Characteristics and Risks of Standardized Options before investing in options.

About Trading API

Trade stocks & crypto with Alpaca's easy to use Trading API. Up to 4X intraday & 2X overnight buying power. Short selling. Advanced order types. All packaged and delivered through our API.

Suggest Edits

Paper Trading

Paper trading is free and available to all Alpaca users. Paper trading is a real-time simulation environment where you can test your code. You can reset and test your algorithm as much as you want using free, real-time market data. For more click here.

Account Plans

Anyone globally can create an paper only account. All you need to do is sign up with your email address. Alpaca also offers live brokerage accounts (with real money) for individuals and businesses plus crypto accounts. For more click here.

Crypto Trading

Alpaca offers crypto trading through our API and the Alpaca web dashboard! Trade all day, seven days a week, as frequently as you'd like. For more click here.

Understand Orders

Our Trading API supports different order types such as market, limit and stop orders plus more complex ones. For more click here.

Fractional Trading

Fractional shares are fractions of a whole share, meaning that you don't need to buy a whole share to own a portion of a company. You can now buy as little as $1 worth of shares for over 2,000 US equities. For more click here.

We only allow market orders for fractional trading at the moment.

User Protections

We have enabled several types of protections to enhance your trading experience. For more click here.

Authentication

How to authenticate to Trading API.

Suggest Edits

Live Trading

Alpaca's live API domain is `https://api.alpaca.markets`.
Every private API call requires key-based authentication. API keys can be acquired in the developer web console. The client must provide a pair of API key ID and secret key in the HTTP request headers named `APCA-API-KEY-ID` and `APCA-API-SECRET-KEY`, respectively.
Here is an example using curl showing how to authenticate with the API.

cURL

```
curl -X GET \
    -H "APCA-API-KEY-ID: {YOUR_API_KEY_ID}"
    -H "APCA-API-SECRET-KEY: {YOUR_API_SECRET_KEY}"
    https://{apiserver_domain}/v2/account
```

Paper Trading

Alpaca's paper trading service uses a different domain and different credentials from the live API. You'll need to connect to the right domain so that you don't run your paper trading algo on your live account.

To use the paper trading api, set `APCA-API-KEY-ID` and `APCA-API-SECRET-KEY` to your paper credentials, and set the domain to `https://paper-api.alpaca.markets`.
After you have tested your algo in the paper environment and are ready to start running your algo in the live environment, you can switch the domain to the live

domain, and the credentials to your live credentials. Your algo will then start trading with real money.

To learn more about paper trading, visit the paper trading page.

Getting Started with Trading API

This section outlines how to install Alpaca's SDKs, create a free alpaca account, locate your API keys, and how to submit orders applicable for both stocks and crypto.

Suggest Edits

Request ID

All trading API endpoint provides a unique identifier of the API call in the response header with `X-Request-ID` key, the Request ID helps us to identify the call chain in our system.

Make sure you provide the Request ID in all support requests that you created, it could help us to solve the issue as soon as possible. Request ID can't be queried in other endpoints, that is why we suggest to persist the recent Request IDs.

Shell

```
$ curl -v https://paper-api.alpaca.markets/v2/account
...
> GET /v2/account HTTP/1.1
> Host: paper-api.alpaca.markets
> User-Agent: curl/7.88.1
> Accept: */*
>
< HTTP/1.1 403 Forbidden
< Date: Fri, 25 Aug 2023 09:34:40 GMT
< Content-Type: application/json
< Content-Length: 26
< Connection: keep-alive
< X-Request-ID: 649c5a79da1ab9cb20742ffdada0a7bb
<
...
```

Working with /account

Lean how to use the /account endpoint to learn about the state of your account

Suggest Edits

View Account Information

By sending a `GET` request to our `/v2/account` endpoint, you can see various information about your account, such as the amount of buying power available or whether or not it has a PDT flag.

PythonJavaScriptC#Go

```
from alpaca.trading.client import TradingClient
```

```
from alpaca.trading.requests import GetAssetsRequest

trading_client = TradingClient('api-key', 'secret-key')

# Get our account information.
account = trading_client.get_account()

# Check if our account is restricted from trading.
if account.trading_blocked:
    print('Account is currently restricted from trading.')

# Check how much money we can use to open new positions.
print('${} is available as buying power.'.format(account.buying_power))
```

View Gain/Loss of Portfolio

You can use the information from the account endpoint to do things like calculating the daily profit or loss of your account.

PythonJavaScriptC#Go

```
from alpaca.trading.client import TradingClient
from alpaca.trading.requests import GetAssetsRequest

trading_client = TradingClient('api-key', 'secret-key')

# Get our account information.
account = trading_client.get_account()

# Check our current balance vs. our balance at the last market close
balance_change = float(account.equity) - float(account.last_equity)
print(f'Today\'s portfolio balance change: ${balance_change}')
```

Working with /assets

Learn how to use the `/assets` endpoint to learn more about assets available on Alpaca. Both Securities and Crypto can be retrieved from the `/assets` endpoint.

Suggest Edits

Get a List of Assets

If you send a `GET` request to our `/v2/assets` endpoint, you'll receive a list of US equities.

PythonJavaScriptC#Go

```
from alpaca.trading.client import TradingClient
from alpaca.trading.requests import GetAssetsRequest
from alpaca.trading.enums import AssetClass

trading_client = TradingClient('api-key', 'secret-key')

# search for US equities
search_params = GetAssetsRequest(asset_class=AssetClass.US_EQUITY)

assets = trading_client.get_all_assets(search_params)
```

See If a Particular Asset is Tradable on Alpaca

By sending a symbol along with our request, we can get the information about just one asset. This is useful if we just want to make sure that a particular asset is tradable before we attempt to buy it.

PythonJavaScriptC#Go

```
from alpaca.trading.client import TradingClient
from alpaca.trading.requests import GetAssetsRequest

trading_client = TradingClient('api-key', 'secret-key')

# search for AAPL
aapl_asset = trading_client.get_asset('AAPL')

if aapl_asset.tradable:
    print('We can trade AAPL.')
```

Working with /orders

Learn how to submit orders to Alpaca.

Suggest Edits

This page contains examples of some of the things you can do with order objects through our API. For additional help understanding different types of orders and how they behave once they're placed, please check out the Orders on Alpaca page.

Place New Orders

Orders can be placed with a `POST` request to our `/v2/orders` endpoint.

PythonJavaScriptC#Go

```
from alpaca.trading.client import TradingClient
from alpaca.trading.requests import MarketOrderRequest
from alpaca.trading.enums import OrderSide, TimeInForce

trading_client = TradingClient('api-key', 'secret-key', paper=True)

# preparing market order
market_order_data = MarketOrderRequest(
                    symbol="SPY",
                    qty=0.023,
                    side=OrderSide.BUY,
                    time_in_force=TimeInForce.DAY
                    )

# Market order
market_order = trading_client.submit_order(
                order_data=market_order_data
```

```
                )

# preparing limit order
limit_order_data = LimitOrderRequest(
                    symbol="BTC/USD",
                    limit_price=17000,
                    notional=4000,
                    side=OrderSide.SELL,
                    time_in_force=TimeInForce.FOK
                   )

# Limit order
limit_order = trading_client.submit_order(
                order_data=limit_order_data
              )
```

Submit Shorts

Short orders can also be placed for securities which you do not hold an open long position in.

PythonC#

```
from alpaca.trading.client import TradingClient
from alpaca.trading.requests import MarketOrderRequest
from alpaca.trading.enums import OrderSide, TimeInForce

trading_client = TradingClient('api-key', 'secret-key', paper=True)

# preparing orders
market_order_data = MarketOrderRequest(
                    symbol="SPY",
                    qty=1,
                    side=OrderSide.SELL,
                    time_in_force=TimeInForce.GTC
                    )

# Market order
market_order = trading_client.submit_order(
                order_data=market_order_data
               )
```

Using Client Order IDs

`client_order_id` can be used to organize and track specific orders in your client program. Unique `client_order_ids` for different strategies is a good way of running parallel algos across the same account.

PythonJavaScriptC#

```
from alpaca.trading.client import TradingClient
from alpaca.trading.requests import MarketOrderRequest
from alpaca.trading.enums import OrderSide, TimeInForce

trading_client = TradingClient('api-key', 'secret-key', paper=True)
```

```
# preparing orders
market_order_data = MarketOrderRequest(
                    symbol="SPY",
                    qty=0.023,
                    side=OrderSide.BUY,
                    time_in_force=TimeInForce.DAY,
                    client_order_id='my_first_order',
                    )

# Market order
market_order = trading_client.submit_order(
                order_data=market_order_data
               )

# Get our order using its Client Order ID.
my_order = trading_client.get_order_by_client_id('my_first_order')
print('Got order #{}'.format(my_order.id))
```

Submitting Bracket Orders

Bracket orders allow you to create a chain of orders that react to execution and stock price. For more details, go to Bracket Order Overview.

PythonJavaScriptC#Go

```
from alpaca.trading.client import TradingClient
from alpaca.trading.requests import MarketOrderRequest, LimitOrderRequest,
TakeProfitRequest, StopLossRequest
from alpaca.trading.enums import OrderSide, TimeInForce, OrderClass

trading_client = TradingClient('api-key', 'secret-key', paper=True)

# preparing bracket order with both stop loss and take profit
bracket__order_data = MarketOrderRequest(
                    symbol="SPY",
                    qty=5,
                    side=OrderSide.BUY,
                    time_in_force=TimeInForce.DAY,
                    order_class=OrderClass.BRACKET,
                    take_profit=TakeProfitRequest(limit_price=400),
                    stop_loss=StopLossRequest(stop_price=300)
                    )

bracket_order = trading_client.submit_order(
                order_data=bracket__order_data
               )

# preparing oto order with stop loss
oto_order_data = LimitOrderRequest(
                    symbol="SPY",
                    qty=5,
                    limit_price=350,
                    side=OrderSide.BUY,
                    time_in_force=TimeInForce.DAY,
                    class=OrderClass.OTO,
                    stop_loss=StopLossRequest(stop_price=300)
```

```
                    )

# Market order
oto_order = trading_client.submit_order(
                order_data=oto_order_data
               )
```

Submitting Trailing Stop Orders

Trailing stop orders allow you to create a stop order that automatically changes the stop price allowing you to maximize your profits while still protecting your position with a stop price. For more details, go to Trailing Stop Order Overview.

PythonJavaScriptC#Go

```
from alpaca.trading.client import TradingClient
from alpaca.trading.requests import TrailingStopOrderRequest
from alpaca.trading.enums import OrderSide, TimeInForce

trading_client = TradingClient('api-key', 'secret-key', paper=True)


trailing_percent_data = TrailingStopOrderRequest(
                    symbol="SPY",
                    qty=1,
                    side=OrderSide.SELL,
                    time_in_force=TimeInForce.GTC,
                    trail_percent=1.00 # hwm * 0.99
                    )

trailing_percent_order = trading_client.submit_order(
                order_data=trailing_percent_data
               )


trailing_price_data = TrailingStopOrderRequest(
                    symbol="SPY",
                    qty=1,
                    side=OrderSide.SELL,
                    time_in_force=TimeInForce.GTC,
                    trail_price=1.00 # hwm - $1.00
                    )

trailing_price_order = trading_client.submit_order(
                order_data=trailing_price_data
               )
```

Retrieve All Orders

If you'd like to see a list of your existing orders, you can send a `GET` request to our `/v2/orders` endpoint.

PythonJavaScriptC#Go

```
from alpaca.trading.client import TradingClient
```

```
from alpaca.trading.requests import GetOrdersRequest
from alpaca.trading.enums import QueryOrderStatus

trading_client = TradingClient('api-key', 'secret-key', paper=True)

# Get the last 100 closed orders
get_orders_data = GetOrdersRequest(
    status=QueryOrderStatus.CLOSED,
    limit=100,
    nested=True  # show nested multi-leg orders
)

trading_client.get_orders(filter=get_orders_data)
```

Listen for Updates to Orders

You can use Websockets to receive real-time updates about the status of your orders as they change. You can see the full documentation here.

PythonJavaScriptC#Go

```
from alpaca.trading.stream import TradingStream

stream = TradingStream('api-key', 'secret-key', paper=True)


@conn.on(client_order_id)
async def on_msg(data):
    # Print the update to the console.
    print("Update for {}. Event: {}.".format(data.event))

stream.subscribe_trade_updates(on_msg)
# Start listening for updates.
stream.run()
```

Working with /positions

Suggest Edits

You can view the positions in your portfolio by making a `GET` request to the `/v2/positions` endpoint. If you specify a symbol, you'll see only your position for the associated stock.

PythonJavaScriptC#Go

```
from alpaca.trading.client import TradingClient

trading_client = TradingClient('api-key', 'secret-key')

# Get our position in AAPL.
aapl_position = trading_client.get_open_position('AAPL')

# Get a list of all of our positions.
portfolio = trading_client.get_all_positions()

# Print the quantity of shares for each position.
for position in portfolio:
    print("{} shares of {}".format(position.qty, position.symbol))
```

The current price reflected will be based on the following:

4:00 am ET - 9:30 am ET - Last trade based on the premarket

9:30 am ET - 4pm ET - Last trade

4:00 pm ET - 10:00 pm ET - Last trade based on after-hours trading

10 pm ET - 4:00 am ET next trading day - Official closing price from the primary exchange at 4 pm ET.

Paper Trading

Test your algos in our paper environment. Free and available to all Alpaca users.

Suggest Edits

Paper trading is a real-time simulation environment where you can test your code. You can reset and test your algorithm as much as you want using free, real-time market data. Paper trading simulates crypto trading as well. Paper trading works the same way as live trading end to end - except the order is not routed a live exchange. Instead, the system simulates the order filling based on the real-time quotes.

When you run your algorithm with the live market, there are many things that can happen that you may not see in backtesting. Orders may not be filled, prices may spike, or your network may get disconnected and retry may be needed. During the software development process, it is important to test your algorithm to catch these things in advance.

Anyone globally can create an Alpaca Paper Only Account! All you need to do is sign up with your email address.

An Alpaca **Paper Only Account** is for paper trading only. It allows you to fully utilize the Alpaca API and run your algorithm in our paper trading environment only. You won't be trading real money, but you will be able to track your simulated activity and balance in the Alpaca web dashboard. As an Alpaca Paper Only Account holder, you are only entitled to receive and make use of IEX market data. For more information about our paper trading environment, please refer to Paper Trading Specification.

Paper vs Live

We are thrilled that many users have found paper trading useful, and we continue to work on improving our paper trading assumptions so that users may have a superior experience. However, please note that paper trading is only a simulation. It provides a good approximation for what one might expect in real trading, but it is not a substitute for real trading and performance may differ. Specifically, paper trading does not account for:

- Market impact of your orders
- Information leakage of your orders
- Price slippage due to latency
- Order queue position (for non-marketable limit orders)
- Price improvement received
- Regulatory fees
- Dividends

If you are interested to incorporate these issues into your testing, you may do so by trading a live brokerage account. Even small amounts of real money can often provide insight into issues not seen in a simulation environment.

Paper vs Live

Feature	Paper	L
Eligibility	✅	✅
API Access	✅	✅
Free IEX Real Time Data	✅	✅
MFA	✅	✅
Margin Trading	✅	✅
Short Selling	✅	✅
Premarket/After Hours Trading	✅	✅
Borrow Fees	⛔ (Coming Soon!)	✅

Comparing Other Simulators

Users may be interested to compare their paper trading results on Alpaca with their paper trading results on other platforms such as Quantopian or Interactive Brokers. Please note there are several factors that may lead to performance differences. Paper trading platforms may have different:

- Fill prices
- Fill assumptions
- Liquidity assumptions
- Return calculation methodologies
- Market data sources

Getting Started with Paper Trading

Your initial paper trading account is created with $100k balance as a default setting. You can reset the paper trading account at any time later with arbitrary amount as you configure.

Your paper trading account will have a different API key from your live account, and all you need to do to start using your paper trading account is to replace your API key and API endpoint with ones for the paper trading. The API spec is the same between the paper trading and live accounts. The API endpoint (base URL) is displayed in your paper trading dashboard, and please follow the instruction about how to set it depending on the library you are using. In most cases, you need to set an environment variable `APCA_API_BASE_URL = https://paper-api.alpaca.markets`

Reset Your Paper Trading Account

You can reset your paper trading account at any time from the dashboard. Go to the paper trading view and push the reset button. Once you reset the paper trading account, the trading history of the paper trading account is wiped out, and a new paper trading account is created based on your requested initial balance. Please make sure to re-generate the API key as the old key is associated with your old paper trading account and is not valid any longer after you reset it.

Rules and Assumptions

- Paper trading account simulates our Pattern Day Trader checks. Orders that would generate a 4th Day Trade within 5 business days

will be rejected if the real-time net worth is below $25,000. Please read the Pattern DayTrader Protection page for more details.

- Paper trading account **does NOT** simulate dividends.
- Paper trading account **does NOT** send order fill emails.
- Market Data API works identically.
- You cannot change the account balance after it is created, unless you reset it.
- Orders are filled only when they become marketable. This means that a non-marketable buy limit order will not be filled until its limit price is equal to or greater than the best ask price, and a non-marketable sell limit order will not be filled until its limit price is equal to or less than the best bid.
- Your order quantity is not checked against the NBBO quantities. In other words, you can submit and receive a fill for an order that is much larger than the actual available liquidity.
  When orders are eligible to be filled, they will receive partial fills for a random size 10% of the time. If the order price is still marketable, the remaining quantity would be re-evaluated for a subsequent fill.
- Regardless of whether you have a Paper Trading Only account or a real money brokerage account, all orders submitted in paper trading will be matched against the best available current market price (NBBO).

Trading Account

- Suggest Edits

Alpaca Brokerage Account (Live Trading)

- After creating an Alpaca Paper Only Account, you can enable live trading by becoming an Alpaca Brokerage Account holder. This requires you to go through an account on-boarding process with Alpaca Securities LLC, a FINRA member and SEC registered broker-dealer. We now support brokerage accounts for individuals and business entities from around the world.

- With a brokerage account, you will be able to fully utilize Alpaca for your automated trading and investing needs. Using the Alpaca API, you'll be able to buy and sell stocks in your brokerage account, and you'll receive real-time consolidated market data. In addition, you will continue to be able to test your strategies and simulate your trades in our paper trading

environment. And with the Alpaca web dashboard, it's easy to monitor both your paper trading and your real money brokerage account. All accounts are opened as margin accounts. Accounts with $2,000 or more equity will have access to margin trading and short selling.

- Individuals
- Alpaca Securities LLC supports individual taxable brokerage accounts. At this time, we do not support retirement accounts.

- Businesses/Incorporated Entities
- You can open a business trading account to use Alpaca for trading purposes, but not for building apps/services.

- 👀
- **Alpaca currently accepts entities that are *Corporations*, *LLCs* and *Partnerships* in the U.S., and around the world. There is a $50,000 minimum deposit required for opening a business account at Alpaca.**

- # Markets Supported
- Currently, Alpaca only supports trading of listed U.S. stocks and select cryptocurrencies.

- # The Account Object
- The account API serves important information related to an account, including account status, funds available for trade, funds available for withdrawal, and various flags relevant to an account's ability to trade.

- An account maybe be blocked for just for trades (`trading_blocked` flag) or for both trades and transfers (`account_blocked` flag) if Alpaca identifies the account to be engaging in any suspicious activity. Also, in accordance with FINRA's pattern day trading rule, an account may be flagged for pattern day trading (`pattern_day_trader` flag), which would inhibit an account from placing any further day-trades.
- Please note that cryptocurrencies are not eligible assets to be used as collateral for margin accounts and will require the asset be traded using cash only.

- Sample Object
- JSON


```
{
  "account_blocked": false,
  "account_number": "010203ABCD",
  "buying_power": "262113.632",
  "cash": "-23140.2",
  "created_at": "2019-06-12T22:47:07.99658Z",
  "currency": "USD",
  "crypto_status": "ACTIVE",
  "non_marginable_buying_power": "7386.56",
  "accrued_fees": "0",
  "pending_transfer_in": "0",
  "pending_transfer_out": "0",
  "daytrade_count": "0",
  "daytrading_buying_power": "262113.632",
  "equity": "103820.56",
  "id": "e6fe16f3-64a4-4921-8928-cadf02f92f98",
  "initial_margin": "63480.38",
  "last_equity": "103529.24",
  "last_maintenance_margin": "38000.832",
  "long_market_value": "126960.76",
  "maintenance_margin": "38088.228",
  "multiplier": "4",
  "pattern_day_trader": false,
  "portfolio_value": "103820.56",
  "regt_buying_power": "80680.36",
  "short_market_value": "0",
  "shorting_enabled": true,
  "sma": "0",
  "status": "ACTIVE",
  "trade_suspended_by_user": false,
  "trading_blocked": false,
  "transfers_blocked": false
}
```


- Account Properties

Attribute	Type	Description
`id`	string	Account ID.
`account_number`	string	Account number.
`status`	string<account_status>	See detailed account statuses below

Attribute	Type	Description
`crypto_status`	string<account_status>	The current status of the crypto enablement. See detailed crypto statuses below.
`currency`	string	"USD"
`cash`	string	Cash balance
`portfolio_value`	string	**[Deprecated]** Total value of cash + holding positions (Equivalent to the equity field)
`non_marginable_buying_power`	string	Current available non-margin dollar buying power
`accrued_fees`	string	The fees collected.
`pending_transfer_in`	string	Cash pending transfer in.
`pending_transfer_out`	string	Cash pending transfer out
`pattern_day_trader`	boolean	Whether or not the account has been flagged as a pattern day trader
`trade_suspended_by_user`	boolean	User setting. If `true`, the account is not allowed to place orders.
`trading_blocked`	boolean	If `true`, the account is not allowed to place orders.
`transfers_blocked`	boolean	If `true`, the account is not allowed to request money transfers.
`account_blocked`	boolean	If `true`, the account activity by user is prohibited.
`created_at`	string	Timestamp this account was created at

Attribute	Type	Description
`shorting_enabled`	boolean	Flag to denote whether or not the account is permitted to short
`long_market_value`	string	Real-time MtM value of all long positions held in the account
`short_market_value`	string	Real-time MtM value of all short positions held in the account
`equity`	string	`cash` + `long_market_value` + `short_market_value`
`last_equity`	string	Equity as of previous trading day at 16:00:00 ET
`multiplier`	string	Buying power (BP) multiplier that represents account margin classification Valid values: - **1** (standard limited margin account with 1x BP), - **2** (reg T margin account with 2x intraday and overnight BP; this is the default for all non-PDT accounts with $2,000 or more equity), - **4** (PDT account with 4x intraday BP and 2x reg T overnight BP)
`buying_power`	string	Current available $ buying power; If multiplier = 4, this is your daytrade buying power which is calculated as (last*equity - (last) maintenance_margin)* 4; If multiplier = 2, buying*power = max(equity – initial_margin,0)* 2; If multiplier = 1, buying_power = cash
`initial_margin`	string	Reg T initial margin requirement (continuously updated value)

Attribute	Type	Description
maintenance_margin	string	Maintenance margin requirement (continuously updated value)
sma	string	Value of special memorandum account (will be used at a later date to provide additional buying_power)
daytrade_count	int	The current number of daytrades that have been made in the last 5 trading days (inclusive of today)
last_maintenance_margin	string	Your maintenance margin requirement on the previous trading day
daytrading_buying_power	string	Your buying power for day trades (continuously updated value)
regt_buying_power	string	Your buying power under Regulation T (your excess equity - equity minus margin value - times your margin multiplier)

- # Account Status ENUMS
- The following are the possible account status values. Most likely, the account status is `ACTIVE` unless there is an issue. The account status may get to `ACCOUNT_UPDATED` when personal information is being updated from the dashboard, in which case you may not be allowed trading for a short period of time until the change is approved.

status	description
ONBOARDING	The account is onboarding.
SUBMISSION_FAILED	The account application submission failed for some reason.
SUBMITTED	The account application has been submitted for review.
ACCOUNT_UPDATED	The account information is being updated.

status	description
APPROVAL_PENDING	The final account approval is pending.
ACTIVE	The account is active for trading.
REJECTED	The account application has been rejected.

Crypto Trading

Trade crypto through our API and the Alpaca web dashboard! Trade all day, seven days a week, as frequently as you'd like.

Suggest Edits

As of November 18, 2022, cryptocurrency trading is open to select international jurisdiction and some U.S. jurisdictions.

To view the supported US regions for crypto trading, click here.

Supported Coins

Alpaca supports over 20+ unique crypto assets across 56 trading pairs. Current trading pairs are based on BTC, USD, USDT and USDC) with more assets and trading pairs coming soon.

To query all available crypto assets and pairs you can you use the following API call:

cURL

```
curl --request GET 'https://api.alpaca.markets/v2/assets?asset_class=crypto' \
--header 'Apca-Api-Key-Id: <KEY>' \
--header 'Apca-Api-Secret-Key: <SECRET>'
```

Below is a sample trading pair object composed of two assets, BTC and USD.

JSON

```
{
  "id": "276e2673-764b-4ab6-a611-caf665ca6340",
  "class": "crypto",
  "exchange": "ALPACA",
  "symbol": "BTC/USD",
  "name": "BTC/USD pair",
  "status": "active",
  "tradable": true,
  "marginable": false,
```

```
  "shortable": false,
  "easy_to_borrow": false,
  "fractionable": true,
  "min_order_size": "0.0001",
  "min_trade_increment": "0.0001",
  "price_increment": "1"
}
```

Note that symbology for trading pairs has changed from our previous format, where `BTC/USD` was previously referred to as `BTCUSD`. Our API has made proper changes to support the legacy convention as well for backwards compatibility. For further reference see Assets API. **add link**

Supported Orders

When submitting crypto orders through the Orders API and the Alpaca web dashboard, Market, Limit and Stop Limit orders are supported while the supported `time_in_force` values are `gtc`, and `ioc`. We accept fractional orders as well with either `notional` or `qty` provided.
You can submit crypto orders for any supported crypto pair via API, see the below cURL POST request.

cURL

```
curl --request POST 'https://paper-api.alpaca.markets/v2/orders' \
--header 'Apca-Api-Key-Id: <KEY>' \
--header 'Apca-Api-Secret-Key: <SECRET>' \
--header 'Content-Type: application/json' \
--data-raw '{
  "symbol": "BTC/USD",
  "qty": "0.0001",
  "side": "buy",
  "type": "market",
  "time_in_force": "gtc"
}'
```

The above request submits a market order via API to buy 0.0001 BTC with USD (BTC/USD pair) that is good till end of day.

To learn more see **orders** and **fractional trading**.

All cryptocurrency assets are fractionable but the supported decimal points vary depending on the cryptocurrency. See **Assets entity** for information on fractional precisions per asset.

Note these values could change in the future.

Crypto Market Data

Alpaca provides free limited crypto data and a more advanced unlimited paid plan. To learn more on our paid plan or to subscribe see **Alpaca Data**.

To request trading pairs data via REST API, see **Crypto Pricing Data REST API Reference**.

The example below requests the latest order book data (bid and asks) for the following three crypto trading pairs: BTC/USD, ETH/BTC and ETH/USD.

cURL

```
curl --request GET
'data.alpaca.markets/v1beta3/crypto/latest/orderbooks?symbols=BTC/USD,ETH/BTC,ETH/
USD' \
--header 'Apca-Api-Key-Id: <KEY>' \
--header 'Apca-Api-Secret-Key: <SECRET>'
```

JSON

```
{
    "orderbooks": {
        "BTC/USD": {
            "a": [
                {
                    "p": 23541,
                    "s": 0.6118
                },
                ...
            ],
            "b": [
                {
                    "p": 23535,
                    "s": 0.0084
                },
                ...
            ],
            "t": "2022-07-19T22:12:18.485428736Z"
        },
        "ETH/USD": { ... }
        },
        "ETH/BTC": { ... }
    }
}
```

Additionally, you can subscribe to real-time crypto data via Websockets. Example below leverages wscat to subscribe to BTC/USD order book.

JSON

```
 $ wscat -c wss://stream.data.alpaca.markets/v1beta2/crypto
Connected (press CTRL+C to quit)
< [{"T":"success","msg":"connected"}]
> {"action": "auth", "key": "<KEY>", "secret": "<SECRET>"}
< [{"T":"success","msg":"authenticated"}]
> {"action":"subscribe","orderbooks":["BTC/USD"]}
<
[{"T":"subscription","trades":[],"quotes":[],"orderbooks":["BTC/USD"],"bars":[],"u
pdatedBars":[],"dailyBars":[]}]
```

< [{"T":"o","S":"BTC/USD","t":"2022-07-19T22:18:43.569566208Z","b":[{"p":23475,"s":0.124},{"p":23474,"s":0.2129},{"...

For further reference of real-time crypto pricing data. see **Real-time Data Reference section.**

Transferring Crypto

Alpaca now offers native on-chain crypto transfers with wallets! If you have crypto trading enabled and reside in an eligible US state or international jurisdiction you can access wallets on the web dashboard via the Crypto Transfers tab.

Alpaca wallets currently support transfers for Bitcoin, Ethereum, and all Ethereum (ERC20) based tokens. To learn more on transferring crypto with Alpaca, see **Crypto Wallets FAQs**

Crypto Fees

While Alpaca stock trading remains commission-free, crypto trading includes a small fee per trade dependent on your executed volume and order type. Any market or exchange consists of two parties, buyers and sellers. When you place an order to buy crypto on the Alpaca Exchange, there is someone else on the other side of the trade selling what you want to buy. The seller's posted order on the order book is providing liquidity to the exchange and allows for the trade to take place. Note, that both buyers and sellers can be makers or takers depending on the order entered and current quote of the coin. **A maker is someone who adds liquidity, and the order gets placed on the order book. A Taker on the other hand removes the liquidity by placing a market or marketable limit order which executes against posted orders.**

See the below table with volume-tiered fee pricing:

Tier	30D Trading Volume (USD)	Maker	Taker
1	0 - 100,000	15 bps	25 bps
2	100,000 - 500,000	12 bps	22 bps
3	500,000 - 1,000,000	10 bps	20 bps
4	1,000,000 - 10,000,000	8 bps	18 bps

Tier	30D Trading Volume (USD)	Maker	Taker
5	10,000,000 - 25,000,000	5 bps	15 bps
6	25,000,000 - 50,000,000	2 bps	13 bps
7	50,000,000 - 100,000,000	2 bps	12 bps
8	100,000,000+	0 bps	10 bps

The crypto fee will be charged on the credited crypto asset/fiat (what you receive) per trade. Some examples,

- Buy `ETH/BTC`, you receive `ETH`, the fee is denominated in `ETH`
- Sell `ETH/BTC`, you receive `BTC`, the fee is denominated in `BTC`
- Buy `ETH/USD`, you receive `ETH`, the fee is denominated in `ETH`
- Sell `ETH/USD`, you receive `USD`, the fee is denominated in `USD`

To get the fees incurred from crypto trading you can use Activities API to query `activity_type` by `CFEE` or `FEE`. See below example of CFEE object:
JSON


```
{
    "id": "20220812000000000::53be51ba-46f9-43de-b81f-576f241dc680",
    "activity_type": "CFEE",
    "date": "2022-08-12",
    "net_amount": "0",
    "description": "Coin Pair Transaction Fee (Non USD)",
    "symbol": "ETHUSD",
    "qty": "-0.000195",
    "price": "1884.5",
    "status": "executed"
}
```


Fees are currently calculated and posted end of day. If you query on same day of trade you might not get results. We will be providing an update for fee posting to be real-time in the near future.

Margin and Short Selling

Cryptocurrencies cannot be bought on margin. This means that you cannot use leverage to buy them and orders are evaluated
against `non_marginable_buying_power`.
Cryptocurrencies can not be sold short.

Trading Hours

Crypto trading is offered for 24 hours everyday and your orders will be executed throughout the day.

Trading Limits

Currently, an order (buy or sell) must not exceed $200k in notional. This is per an order.

Crypto Orders

Suggest Edits

You can submit crypto orders through the traditional orders API endpoints (`POST/orders`).

- The following order types are supported: `market`, `limit` and `stop_limit`
- The following `time_in_force` values are supported: `gtc`, and `ioc`
- We accept fractional orders as well with either `notional` or `qty` provided

You can submit crypto orders for a any supported crypto pair via API, see the below cURL POST request.

cURL

```
curl --request POST 'https://paper-api.alpaca.markets/v2/orders' \
--header 'Apca-Api-Key-Id: <KEY>' \
--header 'Apca-Api-Secret-Key: <SECRET>' \
--header 'Content-Type: application/json' \
--data-raw '{
  "symbol": "BTC/USD",
  "qty": "0.0001",
  "side": "buy",
  "type": "market",
  "time_in_force": "gtc"
}'
```

The above request submits a market order via API to buy 0.0001 BTC with USD (BTC/USD pair) that is good till end of day.

Crypto Pricing Data

Alpaca provides free limited crypto data and a more advanced unlimited paid plan.

Suggest Edits

To request trading pairs data via REST API, see Crypto Pricing Data REST API Reference.

The example below requests the latest order book data (bid and asks) for the following three crypto trading pairs: BTC/USD, ETH/BTC and ETH/USD.

cURL

```
curl --request GET
'https://data.alpaca.markets/v1beta3/crypto/us/latest/orderbooks?symbols=BTC/USD,ETH/BTC,ETH/USD,SOL/USDT' \
--header 'Apca-Api-Key-Id: <KEY>' \
--header 'Apca-Api-Secret-Key: <SECRET>'
```

```
{
    "orderbooks": {
        "BTC/USD": {
            "a": [
                {
                    "p": 27539.494,
                    "s":  0.2632414
                },
                ...
            ],
            "b": [
                {
                    "p": 27511.78083,
                    "s": 0.26265668
                },
                ...
            ],
            "t": "2023-03-18T13:31:44.932988033Z"
        },
        "ETH/USD": { ... },
        "ETH/BTC": { ... },
        "SOL/USDT": { ... }
    }
}
```

Real-Time Crypto Market Data

Additionally, you can subscribe to real-time crypto data via Websockets. Example below leverages wscat to subscribe to:

- BTC/USD trades.
- ETH/USDT and ETH/USD quotes.
- BTC/USD order books

```
 $ wscat -c wss://stream.data.alpaca.markets/v1beta3/crypto/us
Connected (press CTRL+C to quit)
< [{"T":"success","msg":"connected"}]
> {"action": "auth", "key": "<KEY>", "secret": "<SECRET>"}
< [{"T":"success","msg":"authenticated"}]
> {"action":"subscribe", "trades":["BTC/USD"], "quotes":["ETH/USDT","ETH/USD"],
"orderbooks":["BTC/USD"]}
<
[{"T":"subscription","trades":["BTC/USD"],"quotes":["ETH/USDT","ETH/USD"],"orderbo
oks":["BTC/USD"],"updatedBars":[],"dailyBars":[]}]
< [{"T":"o","S":"BTC/USD","t":"2023-03-
18T13:51:29.754747009Z","b":[{"p":27485.3445,"s":0.25893365},{"p":27466.92727,"s":
```

```
0.52351568},...],"a":[{"p":27512.92,"s":0.26137249},{"p":27547.9425,"s":0.52011956
},...],"r":true}]
<
[{"T":"q","S":"ETH/USDT","bp":1815.55510989,"bs":8.24941727,"ap":1818.4,"as":4.151
21428,"t":"2023-03-18T13:51:33.256826818Z"}]
< ...
```


Crypto Fees

Suggest Edits

While Alpaca stock trading remains commission-free, crypto trading includes a small fee per trade dependent on your executed volume and order type. Any market or exchange consists of two parties, buyers and sellers. When you place an order to buy crypto on the Alpaca Exchange, there is someone else on the other side of the trade selling what you want to buy. The seller's posted order on the order book is providing liquidity to the exchange and allows for the trade to take place. Note, that both buyers and sellers can be makers or takers depending on the order entered and current quote of the coin. **A maker is someone who adds liquidity, and the order gets placed on the order book. A Taker on the other hand removes the liquidity by placing a market or marketable limit order which executes against posted orders.**

See the below table with volume-tiered fee pricing:

Tier	30D Crypto Volume (USD)	Maker	Take
1	0 - 100,000	0.15%	0.25%
2	100,000 - 500,000	0.12%	0.22%
3	500,000 - 1,000,000	0.10%	0.20%
4	1,000,000 - 10,000,000	0.08%	0.18%
5	10,000,000- 25,000,000	0.05%	0.15%
6	25,000,000 - 50,000,000	0.02%	0.13%
7	50,000,000 - 100,000,000	0.02%	0.12%
8	100,000,000+	0.00%	0.10%

The crypto fee will be charged on the credited crypto asset/fiat (what you receive) per trade. Some examples:

- Buy ETH/BTC, you receive ETH, the fee is denominated in ETH
- Sell ETH/BTC, you receive BTC, the fee is denominated in BTC
- Buy ETH/USD, you receive ETH, the fee is denominated in ETH
- Sell ETH/USD, you receive USD, the fee is denominated in USD

To get the fees incurred from crypto trading you can use Activities API to query activity_type by CFEE or FEE.
See below example of CFEE object:

JSON


```
{
    "id": "20220812000000000::53be51ba-46f9-43de-b81f-576f241dc680",
    "activity_type": "CFEE",
    "date": "2022-08-12",
    "net_amount": "0",
    "description": "Coin Pair Transaction Fee (Non USD)",
    "symbol": "ETHUSD",
    "qty": "-0.000195",
    "price": "1884.5",
    "status": "executed"
}
```


Fees are currently calculated and posted end of day. If you query on same day of trade you might not get results. We will be providing an update for fee posting to be real-time in the near future.

Check out our Crypto Trading FAQ here

Options Trading

We're excited to support options trading! Use this section to read up on Alpaca's options trading capabilities.

Suggest Edits

Options trading is now available on PAPER!

Share your feedback on Options API for Trading API here

OpenAPI Spec

You can find our Open API docs here: <https://docs.alpaca.markets/reference>.

Enablement

In the Paper environment, options trading capability will be enabled by default - there's nothing you need to do!

Note, in production there will be a more robust experience to request options trading.

For those who do not wish to have options trading enabled, you can disable options trading by navigating to your Trading Dashboard; Account > Configure.

The Trading Account model was updated to reflect the account's options approved and trading levels.

The Account Configuration model was updated to show the maximum options trading level, and allow a user to downgrade to a lower level. Note, a different API will be provided for requesting a level upgrade for live account.

Trading Levels

Alpaca supports the below options trading levels.

Level	Supported Trades	Validation
0	- Options trading is disabled	- NA
1	- Sell a covered call - Sell cash-secured put	- User must own sufficient underlying shares - User must have sufficient options buying power
2	- Level 1 - Buy a call - Buy a put	- User must have sufficient options buying power

Option Contracts

Assets API Update

The Assets endpoint has an existing `attributes` query parameter. Symbols which have option contracts will have an attribute called `options_enabled`.
Querying for symbols with the `options_enabled` attribute allows users to identify the universe of symbols with corresponding option contracts.

Fetch Contracts

To obtain contract details, a new endpoint was introduced: `/v2/options/contracts?underlying_symbols=`. The endpoint supports fetching a single contract such as `/v2/options/contracts/{symbol_or_id}`
The default params are:

- expiration_date_lte: Next weekend

- limit: 100

Example: if /v2/options/contracts is called on Thursday, the response will include Thursday and Friday data. If called on a Saturday, the response will include Saturday, Sunday, Monday, Tuesday, Wednesday, Thursday, and Friday. Here is an example of a response object:

JSON


```
{
    "option_contracts": [
        {
            "id": "6e58f870-fe73-4583-81e4-b9a37892c36f",
            "symbol": "AAPL240119C00100000",
            "name": "AAPL Jan 19 2024 100 Call",
            "status": "active",
            "tradable": true,
            "expiration_date": "2024-01-19",
            "root_symbol": "AAPL",
            "underlying_symbol": "AAPL",
            "underlying_asset_id": "b0b6dd9d-8b9b-48a9-ba46-b9d54906e415",
            "type": "call",
            "style": "american",
            "strike_price": "100",
            "size": "100",
            "open_interest": "6168",
            "open_interest_date": "2024-01-12",
            "close_price": "85.81",
            "close_price_date": "2024-01-12"
        },
...
        ],
   "page_token": "MTAw",
   "limit": 100
}
```


More detailed information regarding this endpoint can be found on the OpenAPI spec here.

Market Data

Alpaca has both streaming data for real-time quotes and trades, and REST API endpoints for the same, however, REST endpoints are considered “historical” as it’s not streaming in real-time and must be specifically called.

OPRA and Indicative Pricing Feed differences

Alpaca provides 2 different feeds to their users:

- OPRA (Options Price Reporting Authority)

- Indicative Pricing Feed

OPRA is the consolidated BBO feed of OPRA. OPRA Plan defines the BBO as the highest bid and lowest offer for a series of options available in one or more of the options markets maintained by the parties. OPRA feed is only available to subscribed users.

Indicative Pricing Feed is a free derivative of the original OPRA feed: the quotes are not actual OPRA quotes, they're just indicative derivatives. The trades are also derivatives and they're delayed by 15 minutes.

Streaming Data

Any users who have Algo Trader Plus subscription and signed the OPRA agreement, can access the OPRA feed. All authenticated users can access our indicative pricing feed for free.

Feed	URL	Quote Subscription Limit	Subsc
OPRA	wss://stream.data.alpaca.markets/v1beta1/opra	1000	Algo Plus
Indicative Pricing	wss://stream.data.alpaca.markets/v1beta1/indicative	200	-

The host part of the URL may change in the future.

Connecting to live streaming data is the same for options as for equities. The process to connect to streaming data is documented here. However, the endpoints are different (see the table above) and **we only support trade and quote subscriptions in MsgPack (binary) format**!

REST Data

Accessing our historical data is similar for options and equities. All users who have signed the OPRA agreement and have Algo Trader Plus subscription can access historical data of the last 15 minutes, the feed parameter is automatically set to `opra` for these users.
Free users can only access data older than 15 minutes and get indicative trades and quotes on the latest endpoints, the feed parameter is implicitly set to `indicative`.
The complete OpenAPI spec can be found here.

Endpoints

Latest Quotes

- Endpoint: /v1beta1/options/quotes/latest?symbols=
- Returns the latest bid and ask prices for the given contract symbols

Latest Trades

- Endpoint: /v1beta1/options/trades/latest
- Returns the latest trade data for the given contract symbols

Snapshots

- Endpoint: /v1beta1/options/snapshots?symbols=
- Returns: Returns the latest trade, and latest quote for the given contract symbols

Option chain

- Endpoint: /v1beta1/options/snapshots/<UNDERLYING>
- Returns all contract symbol snapshots for the given underlying (e.g. AAPL)

Historical Bars

- Endpoint: /v1beta1/options/bars
- Returns the aggregates for the given option symbols between the specified dates

Historical Trades

- Endpoint: /v1beta1/options/trades
- Returns the trades for the given option symbols between the specified dates

Entities

Snapshot Entity

Field	Type	Description
latestTrade	trade entity	Latest trade
latestQuote	quote entity	Latest quote

Trade Entity

Field	Type	Description
t	string[timestamp]	Timestamp in RFC-3339 format with nanosecond precision
p	number	Trade price
s	int	Trade size
x	string	Exchange where the trade occurred
c	string	Trade condition

Quotes Entity

Field	Type	Description
t	string[timestamp]	Timestamp in RFC-3339 format with nanosecond precision
bx	string	Bid exchange
bp	number	Bid price
bs	int	Bid size
ax	string	Ask exchange
ap	number	Ask price
as	int	Ask size
c	string	Quote condition

Placing an Order

Placing an order for an option contract uses the same Orders API that is used for equities and crypto asset classes.

Given the same Orders API endpoint is being used, Alpaca has implemented a series of validations to ensure the options order does not include attributes relevant to other asset classes. Some of these validations include:

- Ensuring `qty` is a whole number

- Notional must not be populated
- time_in_force must be day
- extended_hours must be false or not populated
- type must be one of market, limit, stop, stop_limit

Examples of valid order payloads can be found as a child page here.

Options Positions

Good news - the existing Positions API model will work with options contracts. There is not expected to be a change to this model.

As an additive feature, we aim to support fetching positions of a certain asset class; whether that be options, equities, or crypto.

Account Activities

The existing schema for trade activities (TAs) are applicable for the options asset class. For example, the FILL activity is applicable to options and maintains it's current shape for options.
There are new non-trade activities (NTAs) entry types for options, however the schema stays the same. These NTAs reflect exercise, assignment, and expiry.
More details can be found on a child page here and on the OpenAPI spec here.

🚧
On PAPER NTAs are synced at the start of the following day. While your balance and positions are updated instantly, NTAs on PAPER will be visible in the Activities endpoint only the next day

Exercise and DNE Instructions

Exercise

Contract holders may submit exercise instructions to Alpaca. Alpaca will process instructions and work with our clearing partner accordingly.

All available held shares of this option contract will be exercised. By default, Alpaca will automatically exercise in-the-money (ITM) contracts at expiry.

Exercise requests will be processed immediately once received. Exercise requests submitted outside market hours will be rejected.

Endpoint: `POST /v2/positions/{symbol_or_contract_id}/exercise` (no body)
More details in our OpenAPI Spec here.

Do Not Exercise

To submit a Do-not-exercise (DNE) instruction, please contact our support team.

Expiration

- In the event no instruction is provided on an ITM contract, the Alpaca system will exercise the contract as long as it is ITM by at least $0.01 USD.
- Alpaca Operations has tooling and processes in place to identify accounts which pose a buying power risk with ITM contracts.
- In the event the account does not have sufficient buying power to exercise an ITM position, Alpaca will sell-out the position within 1 hour before expiry.

Assignment

When Alpaca receives an assignment from our clearing partner, we will process the assignment and inform traders via email and websocket streaming events. The assignment websocket streaming event will me modeled as a non-trade activity (NTA).

FAQ

Who has access to options trading?

All users have now access to options trading on Paper.

Options Orders

This page provides examples of valid order payloads

Suggest Edits

Tier 1 Orders

Sell a Covered Call

```
{
  "symbol": "AAPL231201C00195000",
  "qty": "2",
  "side": "sell",
  "type": "limit",
  "limit_price": "1.05",
  "time_in_force": "day"
}
```

Note, the account must have an existing position of 200 shares of AAPL as the order is for 2 contracts, and each option contract is for 100 shares of underlying.

Sell a Cash-Secured Put

```
{
  "symbol": "AAPL231201P00175000",
  "qty": "1",
  "side": "sell",
  "type": "market",
  "time_in_force": "day"
}
```

Note, the account must have sufficient buying power. The account must have ($175 strike) x (100 shares) x (1 contract) = $17,500 USD buying power available.

Tier 2 Orders

Buy a Call

```
{
  "symbol": "AAPL240119C00190000",
  "qty": "1",
  "side": "buy",
  "type": "market",
  "time_in_force": "day"
}
```

The account must have sufficient buying power to afford the call option. If the market order is executed at $5.10, the account must have ($5.10 execution price) x (100 shares) x (1 contract) = $510.00 USD buying power.

Buy a Put

```
{
  "symbol": "AAPL240119P00170000",
  "qty": "1",
  "side": "buy",
  "type": "market",
  "time_in_force": "day"
}
```

The account must have sufficient buying power to afford purchasing put option. If the market order is executed at $1.04, the account must have ($1.04 execution price) x (100 shares) x (1 contract) = $104.00 USD buying power.

Non-Trade Activities for Option Events

This page provides an overview of new NTAs for options-specific events

Suggest Edits

Option Exercise


```
[
  {
    "id": "20190801011955195::5f596936-6f23-4cef-bdf1-3806aae57dbf",
    "activity_type": "OPEXC",
    "date": "2023-07-21",
    "net_amount": "0",
    "description": "Option Exercise",
    "symbol": "AAPL230721C00150000",
    "qty": "-2",
    "status": "executed"
  },
  {
      "id": "20190801011955195::5f596936-6f23-4cef-bdf1-3806aae57dbf",
      "activity_type": "OPTRD",
      "date": "2023-07-21",
      "net_amount": "-30000",
      "description": "Option Trade",
      "symbol": "AAPL",
      "qty": "200",
      "price": "90",
      "status": "executed"
    }
]
```

The exercise event (OPEXC) is applicable to 2 contracts, and the corresponding trade (OPTRD) represents 200 of the underlying shares being purchased at a per-share amount of $150 (strike price).

Option Assignment


```
[
    {
      "id": "20190801011955195::5f596936-6f23-4cef-bdf1-3806aae57dbf",
      "activity_type": "OPASN",
      "date": "2023-07-01",
      "net_amount": "0",
      "description": "Option Assignment",
```

```
    "symbol": "AAPL230721C00150000",
    "qty": "2",
    "status": "executed"
  },
  {
    "activity_type": "OPTRD",
    "id": "20190801011955195::5f596936-6f23-4cef-bdf1-3806aae57dbf",
    "date": "2023-07-01",
    "net_amount": "30000",
    "description": "Option Trade",
    "symbol": "AAPL",
    "qty": "-200",
    "price": "150",
    "status": "executed"
  }
]
```


The assignment event (OPASN) is applicable to 2 contracts, and the corresponding trade (OPTRD) represents 200 of the underlying shares being sold at a per-share amount of $150 (strike price).

ITM Option Expiry

In the event of an in-the-money (ITM) option reaching expiration without being designated as "Do Not Exercise" (DNE), the Alpaca system will automatically initiate the exercise process on behalf of the user. This process mirrors the Exercise event described earlier. In cases where there is insufficient buying power or underlying positions to facilitate the exercise, the system will generate an automated order for the liquidation of the position.

OTM Option Expiry


```
[
  {
    "id": "20190801011955195::5f596936-6f23-4cef-bdf1-3806aae57dbf",
    "activity_type": "OPEXP",
    "date": "2023-07-21",
    "net_amount": "0",
    "description": "Option Expiry",
    "symbol": "AAPL230721C00150000",
    "qty": "-2",
    "status": "executed"
  }
]
```


When a contract expires OTM, the Alpaca system will flatten the position and no further action is taken.

Account Activities

The account activities API provides access to a historical record of transaction activities that have impacted your account.

Suggest Edits

The TradeActivity Object

Sample TradeActivity

JSON


```
{
  "activity_type": "FILL",
  "cum_qty": "1",
  "id": "20190524113406977::8efc7b9a-8b2b-4000-9955-d36e7db0df74",
  "leaves_qty": "0",
  "price": "1.63",
  "qty": "1",
  "side": "buy",
  "symbol": "LPCN",
  "transaction_time": "2019-05-24T15:34:06.977Z",
  "order_id": "904837e3-3b76-47ec-b432-046db621571b",
  "type": "fill"
}
```


Properties

Attribute	Type	Description
`activity_type`	string	For trade activities, this will always be `FILL`
`cum_qty`	string	The cumulative quantity of shares involved in the execution.
`id`	string	An ID for the activity. Always in `::` format. Can be sent as `page_token` in req facilitate the paging of results.
`leaves_qty`	string	For `partially_filled` orders, the quantity of shares that are left to be filled.
`price`	string	The per-share price that the trade was executed at.
`qty`	string	The number of shares involved in the trade execution.
`side`	string	`buy` or `sell`
`symbol`	string	The symbol of the security being traded.
`transaction_time`	string	The time at which the execution occurred.
`order_id`	string	The id for the order that filled.
`type`	string	`fill` or `partial_fill`

The NonTradeActivity (NTA) Object

Sample NTA

JSON


```
{
  "activity_type": "DIV",
  "id": "20190801011955195::5f596936-6f23-4cef-bdf1-3806aae57dbf",
  "date": "2019-08-01",
  "net_amount": "1.02",
  "symbol": "T",
  "qty": "2",
  "per_share_amount": "0.51"
}
```


Properties

Attribute	Type	Description
`activity_type`	string	See below for a list of possible values.
`id`	string	An ID for the activity. Always in `::` format. Can be sent as `page_token` in req facilitate the paging of results.
`date`	string	The date on which the activity occurred or on which the transaction associated activity settled.
`net_amount`	string	The net amount of money (positive or negative) associated with the activity.
`symbol`	string	The symbol of the security involved with the activity. Not present for all activ
`qty`	string	For dividend activities, the number of shares that contributed to the payment. present for other activity types.
`per_share_amount`	string	For dividend activities, the average amount paid per share. Not present for oth activity types.

Pagination of Results

Pagination is handled using the `page_token` and `page_size` parameters. `page_token` represents the ID of the end of your current page of results. If specified with a direction of desc, for example, the results will end before the activity with the specified ID. If specified with a direction of `asc`, results will begin with the activity immediately after the one specified. `page_size` is the maximum number of entries to return in the response. If `date` is not specified, the default and maximum value is 100. If `date` is specified, the default behavior is to return all results, and there is no maximum page size.

Activity Types

activity_type	Description
FILL	Order fills (both partial and full fills)
TRANS	Cash transactions (both CSD and CSW)
MISC	Miscellaneous or rarely used activity types (All types except those in TRANS, DIV, or FI
ACATC	ACATS IN/OUT (Cash)
ACATS	ACATS IN/OUT (Securities)
CFEE	Crypto fee
CSD	Cash deposit(+)
CSW	Cash withdrawal(-)
DIV	Dividends
DIVCGL	Dividend (capital gain long term)
DIVCGS	Dividend (capital gain short term)
DIVFEE	Dividend fee
DIVFT	Dividend adjusted (Foreign Tax Withheld)
DIVNRA	Dividend adjusted (NRA Withheld)
DIVROC	Dividend return of capital
DIVTW	Dividend adjusted (Tefra Withheld)
DIVTXEX	Dividend (tax exempt)
FEE	Fee denominated in USD
INT	Interest (credit/margin)
INTNRA	Interest adjusted (NRA Withheld)
INTTW	Interest adjusted (Tefra Withheld)
JNL	Journal entry
JNLC	Journal entry (cash)
JNLS	Journal entry (stock)
MA	Merger/Acquisition

activity_type	Description
NC	Name change
OPASN	Option assignment
OPEXP	Option expiration
OPXRC	Option exercise
PTC	Pass Thru Charge
PTR	Pass Thru Rebate
REORG	Reorg CA
SC	Symbol change
SSO	Stock spinoff
SSP	Stock split

Fractional Trading

Suggest Edits

Fractional shares are fractions of a whole share, meaning that you don't need to buy a whole share to own a portion of a company. You can now buy as little as $1 worth of shares for over 2,000 US equities.

By default all Alpaca accounts are allowed to trade fractional shares in both live and paper environments. Please make sure you reset your paper account if you run into any issues dealing with fractional shares.

Supported Order Types

Alpaca currently supports fractional trading for market and day order types (support for other order types are in our roadmap). You can pass either a fractional amount (qty), or a notional value (notional) in any POST/v2/orders request. Note that entering a value for either parameters, will automatically nullify the other. If both qty and notional are entered the request will be rejected with an error status 400.

Both notional and qty fields can take up to 9 decimal point values.

Sample Requests

Notional Request

JSON

```
{
  "symbol": "AAPL",
  "notional": 500.75,
  "side": "buy",
  "type": "market",
  "time_in_force": "day"
}
```

Fractional Request

JSON

```
{
  "symbol": "AAPL",
  "qty": 3.654,
  "side": "buy",
  "type": "market",
  "time_in_force": "day"
}
```

Supported Assets

Not all assets are fractionable yet so please make sure you query assets details to check for the parameter `fractionable = true`.
Supported fractionable assets would return a response that looks like this

JSON

```
{
  "id": "b0b6dd9d-8b9b-48a9-ba46-b9d54906e415",
  "class": "us_equity",
  "exchange": "NASDAQ",
  "symbol": "AAPL",
  "name": "Apple Inc. Common Stock",
  "status": "active",
  "tradable": true,
  "marginable": true,
  "shortable": true,
  "easy_to_borrow": true,
  "fractionable": true
}
```

If you request a fractional share order for a stock that is not yet fractionable, the order will get rejected with an error message that reads `requested asset is not fractionable`.

Dividends

Dividend payments occur the same way in fractional shares as with whole shares, respecting the proportional value of the share that you own.

For example if the dividend amount is $0.10 per share and you own 0.5 shares of that stock then you will receive $0.05 as dividend. As a general rule of thumb all dividends are rounded to the nearest penny.

Notes on Fractional Trading

- We do not support short sales in fractional orders. All fractional sell orders are marked long.
- The expected price of fill is the NBBO quote at the time the order was submitted. If you submit an order for a whole and fraction, the price for the whole share fill will be used to price the fractional portion of the order.
- Day trading fractional shares counts towards your day trade count.
- You can cancel a fractional share order that is pending, the same way as whole share orders.
- Limit orders are supported for both fractional and notional orders. Extended hours are also supported with limit orders (same as whole share orders).
- Fees for fractional trading work the same way as with whole shares.

Alpaca does not make recommendations with regard to fractional share trading, whether to use fractional shares at all, or whether to invest in any specific security. A security's eligibility on the list of fractional shares available for trading is not an endorsement of any of the securities, nor is it intended to convey that such stocks have low risk. Fractional share transactions are executed either on a principal or riskless principal basis, and can only be bought or sold with market orders during normal market hours.

Margin and Short Selling

Suggest Edits

In order to trade on margin or sell short, you must have $2,000 or more account equity. Accounts with less than $2,000 will not have access to these features and will be restricted to 1x buying power.

This is only for Equities Trading. Margin Trading for Crypto is not applicable. In addition, PDT checks do not count towards crypto orders or fills

How Margin Works

Trading on margin allows you to trade and hold securities with a value that exceeds your account equity. This is made possible by funds loaned to you by your broker, who uses your account's cash and securities as collateral. For example, a Reg T Margin Account holding $10,000 cash may purchase and hold up to $20,000 in marginable securities overnight (Note: some securities may have a higher maintenance margin requirement making the full 2x overnight buying power effectively unavailable). In addition to the 2x buying power afforded to margin accounts, a Reg T Margin Account flagged as a Pattern Day Trader(PDT) with $25,000 or greater equity will further be allowed to use up to 4x intraday buying power. As an example, a PDT account holding $50,000 cash may purchase and hold up to $200,000 in securities intraday; however, to avoid receiving a margin call the next morning, the securities held would need to be reduced to $100,000 or less depending on the maintenance margin requirement by the end of the day.

Initial Margin

Initial margin denotes the percentage of the trade price of a security or basket of securities that an account holder must pay for with available cash in the margin account, additions to cash in the margin account or other marginable securities.

Alpaca applies a minimum initial margin requirement of 50% for marginable securities and 100% for non-marginable securities per Regulation T of the Federal Reserve Board.

Maintenance Margin

Maintenance margin is the amount of cash or marginable securities required to continue holding an open position. FINRA has set the minimum maintenance requirement to at least 25% of the total market value of the securities, but brokers are free to set higher requirements as part of their risk management.

Alpaca uses the following table to calculate the overnight maintenance margin applied to each security held in an account:

Position Side	Condition	Margin Requirement
LONG	share price < $2.50	100% of EOD market value

Position Side	Condition	Margin Requirement
LONG	share price >= $2.50	30% of EOD market value
LONG	2x Leveraged ETF	100% of EOD market value
LONG	3x Leveraged ETF	100% of EOD market value
SHORT	share price < $5.00	Greater of $2.50/share or 100%
SHORT	share price >= $5.00	Greater of $5.00/share or 30%

Margin Calls

If your account does not satisfy its initial and maintenance margin requirements at the end of the day, you will receive a margin call the following morning. We will contact you and advise you of the call amount that you will need to satisfy either by depositing new funds or liquidating some or all of your positions to reduce your margin requirement sufficiently.

We may contact you prior to the end of the day and ask you to liquidate your positions immediately in the event that your account equity is materially below your maintenance requirement. Furthermore, although we will make every effort to contact you so that you can determine how to best resolve your margin call, we reserve the right to liquidate your holdings in the event we cannot get ahold of you and your account equity is in danger of turning negative.

Calculating and tracking your margin requirement at all times is helpful to avoid receiving a margin call. We strongly recommend doing so if you plan to aggressively use overnight leverage. Please use a 50% initial requirement and refer to the maintenance margin table above. In the future, we will provide real-time estimated initial and maintenance margin values as part of the Account API to help users better manage their risk.

Margin Interest Rate

We are pleased to offer a competitive and low annual margin interest rate of 8.5% (check “Alpaca Securities Brokerage Fee Schedule” on **Important Disclosures** for the most up to dated rate).

The rate is charged only on your account’s end of day (overnight) debit balance using the following calculation:

```
daily_margin_interest_charge = (settlement_date_debit_balance * 0.085) / 360
```

Interest will accrue daily and post to your account at the end of each month. Note that if you have a settlement date debit balance as of the end of day Friday, you will incur interest charges for 3 days (Fri, Sat, Sun).

As an example, if you deposited $10,000 into your account and bought $15,000 worth of securities that you held at the end of the day, you would be borrowing $5,000 overnight and would incur a daily interest expense of ($5000 * 0.085) / 360 = $1.05.

On the other hand, if you deposited $10,000 and bought $15,000 worth of stock that you liquidated the same day, you would not incur any interest expense. In other words, this allows you to make use of the additional buying power for intraday trading without any cost.

Stock Borrow Rates

Alpaca currently only supports opening short positions in easy to borrow ("ETB") securities. Any open short order in a stock that changes from ETB to HTB overnight will be automatically cancelled prior to market open.

Alpaca passes through all borrow costs incurred when a customer shorts a stock. Borrow fees accrue daily and are billed at the end of each month. Borrow fees can vary significantly depending upon demand to short. Generally, ETBs cost between 30 and 300bps annually.

Please note that stock borrow availability changes daily, and we update our assets table each morning, so please use our API to check each stock's borrow status daily. It is infrequent but names can go from ETB → HTB and vice versa.

While we do not currently support opening short positions in hard to borrow ("HTB") securities, we will not force you to close out a position in a stock that has gone from ETB to HTB unless the lender has called the stock. If a stock you hold short has gone from ETB to HTB, you will incur a higher daily stock borrow fee for that stock. We do not currently provide HTB rates via our API, so please contact us in these cases.

Daily stock borrow fees are the fees incurred for all ETB shorts held in your account as of end of day plus any HTB shorts held at any point during the day, calculated as:

```
Daily stock borrow fee = Daily ETB stock borrow fee + Daily HTB stock borrow fee
```

Where,

```
Daily ETB stock borrow fee = (settlement date end of day total ETB short $ market value * that stock's ETB rate) / 360
```

And,

```
Daily HTB stock borrow fee = Σ((each stock's HTB short $ market value * that stock's HTB rate) / 360)
```

Please note that if you hold short positions as of a Friday settlement date, you will incur stock borrow fees for 3 days (Friday, Saturday and Sunday).

Stock borrow fees are charged in the nearest round lot (100 shares) regardless of how many shares were actually shorted. This is because stocks are borrowed in round lots.

Orders at Alpaca

Suggest Edits

Using Alpaca Trade API, a user can monitor, place and cancel their orders with Alpaca. Each order has a unique identifier provided by the client. This client-side unique order ID will be automatically generated by the system if not provided by the client, and will be returned as part of the order object along with the rest of the fields described below. Once an order is placed, it can be queried using the client-side order ID or system-assigned unique ID to check the status. Updates on open orders at Alpaca will also be sent over the streaming interface, which is the recommended method of maintaining order state.

Buying Power

In order to accept your orders that would open new positions or add to existing ones, your account must have sufficient buying power. Alpaca applies a "buying" power check to both buy long and sell short positions.

The calculated value of an opening sell short order is MAX(order's limit price, 3% above the current ask price) multiplied by the order's quantity. In the case of market short orders, the value is simply 3% above the current ask price * order quantity.

The order's calculated value is then checked against your available buying power to determine if it can be accepted. Please note that your available buying power is reduced by your existing open buy long and sell short orders, whereas your sell long and buy to cover orders do not replenish your available buying power until they have executed.

For example, if your buying power is $10,000 and you submit a limit buy order with an order value of $3,000, your order will be accepted and your remaining

available buying power will be $7,000. Even if this order is unfilled, as long as it is open and has not been cancelled, it will count against your available buying power. If you then submitted another order with an order value of $8,000, it would be rejected.

Initial buying power checks on orders:

When the core session is open: Far side of the NBBO (using Bid & Ask Price)

If the order is entered while the extended hours session is open: Midpoint of the inside market

If the order is entered when the core session and extended hours session are closed (pre-market hours): the latest trade from market cache

Equities Trading

The following section deals with equity trading

Orders Submitted Outside of Eligible Trading Hours

Orders not eligible for extended hours submitted after 4:00pm ET will be queued up for release the next trading day.

Orders eligible for extended hours submitted outside of 4:00am - 8:00pm ET are handled as described in the section below. Fractional shares and orders with notional sizes are not available during extended hours trading.

Extended Hours Trading

Using API v2, you can submit and fill orders during pre-market and after-hours. Extended hours trading has specific risks due to the less liquidity. Please read through Alpaca's Extended Hours Trading Risk Disclosure for more details.

Currently, we support full extended hours:

- Pre-market: 4:00am - 9:30am ET, Monday to Friday
- After-hours: 4:00pm - 8:00pm ET, Monday to Friday

Submitting an Extended Hours Eligible Order

To indicate an order is eligible for extended hours trading, you need to supply a boolean parameter named `extended_hours` to your order request. By setting this parameter to `true`, the order is will be eligible to execute in the pre-market or after-hours.
Only limit day orders will be accepted as extended hours eligible. All other order types and TIFs will be rejected with an error. You must adhere to these settings in order to participate in extended hours:

1. The order type must be set to `limit` (with a limit price). Any other type of orders will be rejected with an error.
2. Time-in-force must be set to be `day`. Any other `time_in_force` will be rejected with an error.

For Fractional Orders, starting on October 27, 2021, customers will not be able to send the following sequence of orders outside of market hours for the same security. This is so customers will not have a net short position. Examples of the order sequences that will be rejected are shown below.

Summary	Order 1	Order 2	Second Order Handling Accept/
2x Sell	Notional Sell	Quantity Sell	Reject
2x Sell	Notional Sell	Notional Sell	Reject
2x Sell	Quantity Sell	Notional Sell	Reject
2 x Sell [with Correct Quantity]	Quantity Sell	Quantity Sell	Accept
2 x Sell [with Correct Quantity]	Quantity Sell	Quantity Sell	Reject

For more information - please see our Blog Post on this topic.

If this is done you will see the following error message: `"unable to open new notional orders while having open closing position orders"`.
All symbols supported during regular market hours are also supported during extended hours. Short selling is also treated the same.

IPO Symbols

Alpaca supports trading symbols which have recently begun trading publicly on major exchanges such as the NYSE and NASDAQ. This process is known as a company going public, or an IPO.

As a registered broker/dealer, Alpaca must follow FINRA regulations regarding order types for IPO symbols:

- Alpaca is only able to accept `limit` orders prior to the security's first trade on the exchange
- Once an IPO begins trading on an exchange, `market` orders are accepted

To assist our customers, Alpaca will expose an `attribute` called `ipo` on the Assets model. The `ipo` attribute will flag to customers that this particular symbol has an IPO coming up, or is just about to begin trading on an exchange, and therefore a `limit` order must be used.

Order Types

When you submit an order, you can choose one of supported order types. Currently, Alpaca supports four different types of orders.

Market Order

A market order is a request to buy or sell a security at the currently available market price. It provides the most likely method of filling an order. Market orders fill nearly instantaneously.

As a trade-off, your fill price may slip depending on the available liquidity at each price level as well as any price moves that may occur while your order is being routed to its execution venue. There is also the risk with market orders that they may get filled at unexpected prices due to short-term price spikes.

Limit Order

A limit order is an order to buy or sell at a specified price or better. A buy limit order (a limit order to buy) is executed at the specified limit price or lower (i.e., better). Conversely, a sell limit order (a limit order to sell) is executed at the specified limit price or higher (better). Unlike a market order, you have to specify the limit price parameter when submitting your order.

While a limit order can prevent slippage, it may not be filled for a quite a bit of time, if at all. For a buy limit order, if the market price is within your specified limit price, you can expect the order to be filled. If the market price is equivalent to your limit price, your order may or may not be filled; if the order cannot immediately execute against resting liquidity, then it is deemed non-marketable and will only be filled once a marketable order interacts with it. You could miss a

trading opportunity if price moves away from the limit price before your order can be filled.

Sub-penny increments for limit orders

The minimum price variance exists for limit orders. Orders received in excess of the minimum price variance will be rejected.

- Limit price >=$1.00: Max Decimals= 2
- Limit price <$1.00: Max Decimals = 4

```
{
  "code": 42210000,
  "message": "invalid limit_price 290.123. sub-penny increment does not fulfill
minimum pricing criteria"
}
```

Stop Orders

A stop (market) order is an order to buy or sell a security when its price moves past a particular point, ensuring a higher probability of achieving a predetermined entry or exit price. Once the order is elected, the stop order becomes a market order. Alpaca converts buy stop orders into stop limit orders with a limit price that is 4% higher than a stop price < $50 (or 2.5% higher than a stop price >= $50). Sell stop orders are not converted into stop limit orders.

A stop order does not guarantee the order will be filled at a certain price after it is converted to a market order.

In order to submit a stop order, you will need to specify the stop price parameter in the API.

Sub-penny increments for stop orders

The minimum price variance exists for stop orders. Orders received in excess of the minimum price variance will be rejected.

- Stop price >=$1.00: Max Decimals= 2
- Stop price <$1.00: Max Decimals = 4

```
{
  "code": 42210000,
  "message": "invalid stop_price 290.123. sub-penny increment does not fulfill
minimum pricing criteria"
}
```

Stop Limit Order

A stop-limit order is a conditional trade over a set time frame that combines the features of a stop order with those of a limit order and is used to mitigate risk. The stop-limit order will be executed at a specified limit price, or better, after a given stop price has been reached. Once the stop price is reached, the stop-limit order becomes a limit order to buy or sell at the limit price or better. In the case of a gap down in the market that causes the election of your order, but not the execution, you order will remain active as a limit order until it is executable or cancelled.

In order to submit a stop limit order, you will need to specify both the limit and stop price parameters in the API.

Opening and Closing Auction Orders

Market on open and limit on open orders are only eligible to execute in the opening auction. Market on close and limit on close orders are only eligible to execute in the closing auction. Please see the Time in Force section for more details.

Bracket Orders

A bracket order is a chain of three orders that can be used to manage your position entry and exit. It is a common use case of an OTOCO (One Triggers OCO {One Cancels Other}) order.

The first order is used to enter a new long or short position, and once it is completely filled, two conditional exit orders are activated. One of the two closing orders is called a take-profit order, which is a limit order, and the other is called a stop-loss order, which is either a stop or stop-limit order. Importantly, only one of the two exit orders can be executed. Once one of the exit orders is filled, the other is canceled. Please note, however, that in extremely volatile and fast market conditions, both orders may fill before the cancellation occurs.

Without a bracket order, you would not be able to submit both entry and exit orders simultaneously since Alpaca's system only accepts exit orders for existing positions. Additionally, even if you had an open position, you would not be able to submit two conditional closing orders since Alpaca's system would view one of the two orders as exceeding the available position quantity. Bracket orders

address both of these issues, as Alpaca's system recognizes the entry and exit orders as a group and queues them for execution appropriately.

In order to submit a bracket order, you need to supply additional parameters to the API. First, add a parameter `order_class` as "bracket". Second, give two additional fields `take_profit` and stop_loss both of which are nested JSON objects. The `take_profit` object needs `limit_price` as a field value that specifies limit price of the take-profit order, and the `stop_loss` object needs a mandatory `stop_price` field and optional `limit_price` field. If `limit_price` is specified in `stop_loss`, the stop-loss order is queued as a stop-limit order, but otherwise it is queued as a stop order.
An example JSON body parameter to submit a bracket order is as follows.

JSON

```
{
  "side": "buy",
  "symbol": "SPY",
  "type": "market",
  "qty": "100",
  "time_in_force": "gtc",
  "order_class": "bracket",
  "take_profit": {
    "limit_price": "301"
  },
  "stop_loss": {
    "stop_price": "299",
    "limit_price": "298.5"
  }
}
```

This creates three orders.

- A buy market order for 100 SPY with GTC
- A sell limit order for the same 100 SPY, with limit price = 301
- A sell stop-limit order, with stop price = 299 and limit price = 298.5

The second and third orders won't be active until the first order is completely filled. Additional bracket order details include:

- If any one of the orders is canceled, any remaining open order in the group is canceled.
- `take_profit.limit_price` must be higher than `stop_loss.stop_price` for a buy bracket order, and vice versa for a sell.
- Both `take_profit.limit_price` and `stop_loss.stop_price` must be present.
- Extended hours are not supported. `extended_hours` must be "false" or omitted.

- `time_in_force` must be `day` or `gtc`.
- Each order in the group is always sent with a DNR/DNC (Do Not Reduce/Do Not Cancel) instruction. Therefore, the order price will not be adjusted and the order will not be canceled in the event of a dividend or other corporate action.
- If the take-profit order is partially filled, the stop-loss order will be adjusted to the remaining quantity.
- Order replacement (`PATCH /v2/orders`) is supported to update `limit_price` and `stop_price`.

Each order of the group is reported as an independent order in GET /v2/orders endpoint. But if you specify additional parameter nested=true, the order response will nest the result to include child orders under the parent order with an array field legs in the order entity.

OCO Orders

OCO (One-Cancels-Other) is another type of advanced order type. This is a set of two orders with the same side (buy/buy or sell/sell) and currently only exit order is supported. In other words, this is the second part of the bracket orders where the entry order is already filled, and you can submit the take-profit and stop-loss in one order submission.

With OCO orders, you can add take-profit and stop-loss after you open the position, without thinking about those two legs upfront.

In order to submit an OCO order, specify “oco” for the order_class parameter.

SELLBUY

```
{
  "side": "sell",
  "symbol": "SPY",
  "type": "limit",
  "qty": "100",
  "time_in_force": "gtc",
  "order_class": "oco",
  "take_profit": {
    "limit_price": "301"
  },
  "stop_loss": {
    "stop_price": "299",
    "limit_price": "298.5"
  }
}
```

The type parameter must always be “limit”, indicating the take-profit order type is a limit order. The stop-loss order is a stop order if only `stop_price` is specified, and is a stop-limit order if both `limit_price` and `stop_price` are specified (i.e. `stop_price` must be present in any case). Those two orders work exactly the same way as the two legs of the bracket orders.
Note that when you retrieve the list of orders with the `nested` parameter true, the take-profit order shows up as the parent order while the stop-loss order appears as a child order.
Like bracket orders, order replacement is supported to update limit_price and stop_price.

OTO Orders

OTO (One-Triggers-Other) is a variant of bracket order. It takes one of the take-profit or stop-loss order in addition to the entry order. For example, if you want to set only a stop-loss order attached to the position, without a take-profit, you may want to consider OTO orders.

The order submission is done with the `order_class` parameter be “oto”.
JSON

```
{
  "side": "buy",
  "symbol": "SPY",
  "type": "market",
  "qty": "100",
  "time_in_force": "gtc",
  "order_class": "oto",
  "stop_loss": {
    "stop_price": "299",
    "limit_price": "298.5"
  }
}
```

Either of `take_profit` or `stop_loss` must be present (the above example is for take-profit case), and the rest of requirements are the same as the bracket orders.
Like bracket orders, order replacement is not supported yet.

Threshold on stop price of stop-loss orders

For the stop-loss order leg of advanced orders, please be aware the order request can be rejected because of the restriction of the stop_price parameter value. The stop price input has to be at least $0.01 below (for stop-loss sell, above for buy) than the “base price”. The base price is determined as follows.

- It is the limit price of the take-profit, for OCO orders.

- It is the limit price of the entry order, for bracket or OTO orders if the entry type is limit.
- It is also the current market price for any, of OCO, OTO and bracket.

This restriction is to avoid potential race-conditions in the order handling, but as we improve our system capability, this may be loosened in the future.

Trailing Stop Orders

Trailing stop orders allow you to continuously and automatically keep updating the stop price threshold based on the stock price movement. You request a single order with a dollar offset value or percentage value as the trail and the actual stop price for this order changes as the stock price moves in your favorable way, or stay at the last level otherwise. This way, you don't need to monitor the price movement and keep sending replace requests to update the stop price close to the latest market movement.

Trailing stop orders keep track of the highest (for sell, lowest for buy) prices (called high water mark, or hwm) since the order was submitted, and the user-specified trail parameters determine the actual stop price to trigger relative to high water mark. Once the stop price is triggered, the order turns into a market order, and it may fill above or below the stop trigger price.

To submit a trailing stop order, you will set the type parameter to "trailing_stop". There are two order submission parameters related to trailing stop, one of which is required when type is "trailing_stop".

field	type	description
trail_price	string	a dollar value away from the highest water mark. If you set this to 2.00 for a sell trailing stop, the stop price is always `hwm - 2.00`.
trail_percent	string	a percent value away from the highest water mark. If you set this to 1.0 for a sell trailing stop, the stop price is always `hwm \* 0.99`.

One of these values must be set for trailing stop orders. The following is an example of trailing order submission JSON parameter.

JSON

```
{
  "side": "sell",
  "symbol": "SPY",
  "type": "trailing_stop",
```

```
  "qty": "100",
  "time_in_force": "day",
  "trail_price": "6.15"
}
```

The Order entity returned from the `GET` method has a few fields related to trailing stop orders.

field	type	description
`trail_price`	string	This is the same value as specified when the order was submitted. It will be null if this was not specified.
`trail_percent`	string	This is the same value as specified when the order was submitted. It will be null if this was not specified.
`hwm`	string	The high water mark value. This continuously changes as the market moves towards your favorable way.
`stop_price`	string	This is the same as stop price in the regular stop/stop limit orders, but this is derived from `hwm` and trail parameter, and continuously updates as hwm changes.

If a trailing stop order is accepted, the order status becomes “new”. While the order is pending stop price trigger, you can update the trail parameter by the PATCH method.

field	type	description
`trail`	string	The new value of the `trail_price` or `trail_percent` value. Such a replace request is effective only for the order type is “trailing_stop” before the stop price is hit. Note, you cannot change the price trailing to the percent trailing or vice versa.

Some notes on trailing stop orders

- Trailing stop will not trigger outside of the regular market hours.
- Valid time-in-force values for trailing stop are “day” and “gtc”.
- Trailing stop orders are currently supported only with single orders. However, we plan to support trailing stop as the stop loss leg of bracket/OCO orders in the future.

Proper use of Trailing Stop orders requires understanding the purpose and how they operate. The primary point to keep in mind with Trailing Stop orders is to ensure the difference between the trailing stop and the price is big enough that typical price fluctuations do not trigger a premature execution.

In fast moving markets, the execution price may be less favorable than the stop price. The potential for such vulnerability increases for GTC orders across

trading sessions or stocks experiencing trading halts. The stop price triggers a market order and therefore, it is not necessarily the case that the stop price will be the same as the execution price.

With regard to stock splits, Alpaca reserves the right to cancel or adjust pricing and/or share quantities of trailing stop orders based upon its own discretion. Since Alpaca relies on third parties for market data, corporate actions or incorrect price data may cause a trailing stop to be triggered prematurely.

Time in Force

Crypto Time in Force

For Crypto Trading, Alpaca only supports `gtc`, and `ioc`.
`OPG`, `fok`, `day`, and `CLS` are not supported.

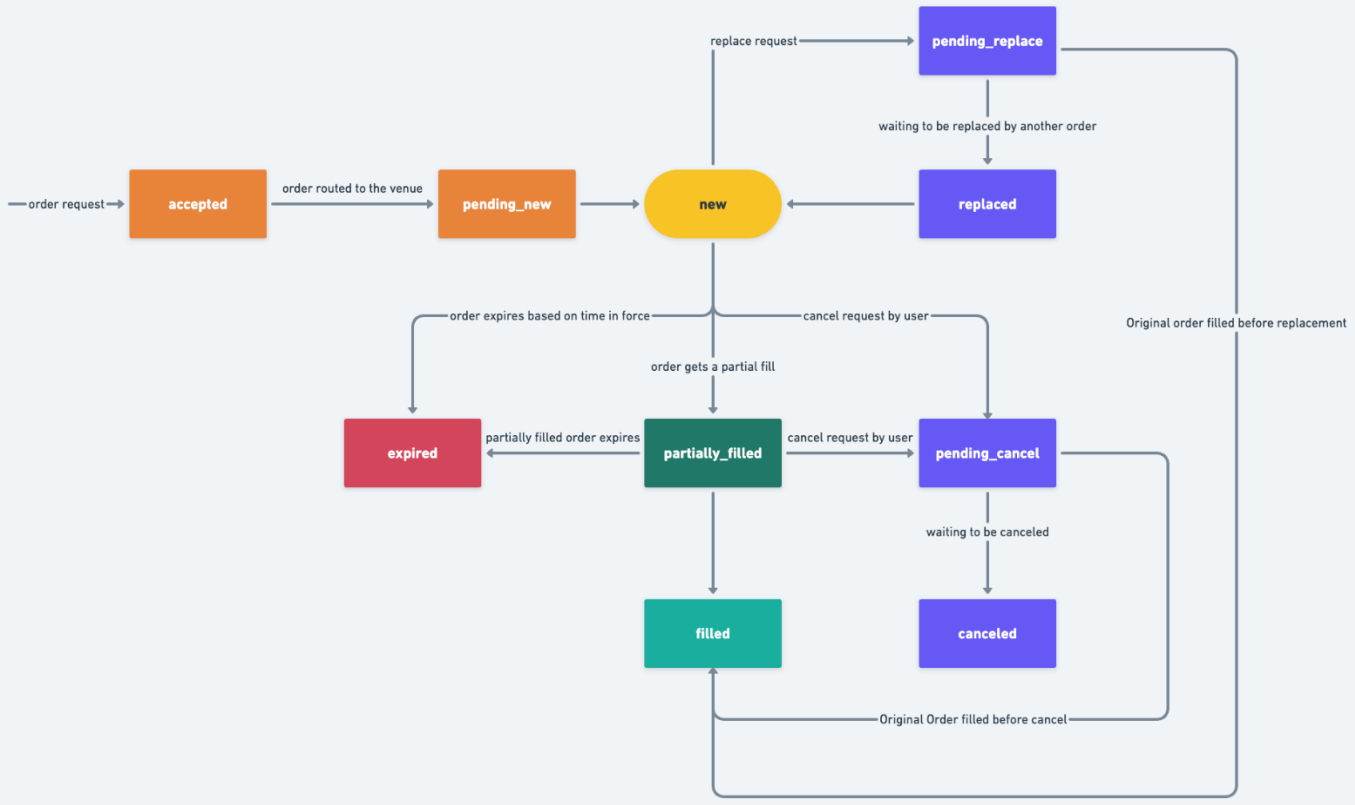
Alpaca supports the following Time-In-Force designations:

time_in_force	description
`day`	A day order is eligible for execution only on the day it is live. By default, the order is only valid during Regular Trading Hours (9:30am - 4:00pm ET). If unfilled after the closing auction, it is automatically canceled. If submitted after the close, it is queued and submitted the following trading day. However, if marked as eligible for extended hours, the order can also execute during supported extended hours.
`gtc`	The order is good until canceled. Non-marketable GTC limit orders are subject to price adjustments to offset corporate actions affecting the issue. We do not currently support Do Not Reduce(DNR) orders to opt out of such price adjustments.
`opg`	Use this TIF with a market/limit order type to submit “market on open” (MOO) and “limit on open” (LOO) orders. This order is eligible to execute only in the market opening auction. Any unfilled orders after the open will be cancelled. OPG orders submitted after 9:28am but before 7:00pm ET will be rejected. OPG orders submitted after 7:00pm will be queued and routed to the following day’s opening auction. On open/on close orders are routed to the primary exchange. Such orders do not necessarily execute exactly at 9:30am / 4:00pm ET but execute per the exchange’s auction rules.

time_in_force	description
`cls`	Use this TIF with a market/limit order type to submit “market on close” (MOC) and “limit on close” (LOC) orders. This order is eligible to execute only in the market closing auction. Any unfilled orders after the close will be cancelled. CLS orders submitted after 3:50pm but before 7:00pm ET will be rejected. CLS orders submitted after 7:00pm will be queued and routed to the following day’s closing auction. Only available with API v2.
`ioc`	An Immediate Or Cancel (IOC) order requires all or part of the order to be executed immediately. Any unfilled portion of the order is canceled. Only available with API v2. Most market makers who receive IOC orders will attempt to fill the order on a principal basis only, and cancel any unfilled balance. On occasion, this can result in the entire order being cancelled if the market maker does not have any existing inventory of the security in question.
`fok`	A Fill or Kill (FOK) order is only executed if the entire order quantity can be filled, otherwise the order is canceled. Only available with API v2.

Order Lifecycle

An order executed through Alpaca can experience several status changes during its lifecycle.



The most common statuses are described in detail below:

status	description
`new`	The order has been received by Alpaca, and routed to exchanges for execution. This is the usual initial state of an order.
`partially_filled`	The order has been partially filled.
`filled`	The order has been filled, and no further updates will occur for the order.
`done_for_day`	The order is done executing for the day, and will not receive further updates until the next trading day.
`canceled`	The order has been canceled, and no further updates will occur for the order. This can be either due to a cancel request by the user, or the order has been canceled by the exchanges due to its time-in-force.
`expired`	The order has expired, and no further updates will occur for the order.
`replaced`	The order was replaced by another order, or was updated due to a market event such as corporate action.

status	description
`pending_cancel`	The order is waiting to be canceled.
`pending_replace`	The order is waiting to be replaced by another order. The order will reject cancel request while in this state.

Less common states are described below. Note that these states only occur on very rare occasions, and most users will likely never see their orders reach these states:

status	description
`accepted`	The order has been received by Alpaca, but hasn't yet been routed to the execution venue. This could be seen often out side of trading session hours.
`pending_new`	The order has been received by Alpaca, and routed to the exchanges, but has not yet been accepted for execution. This state only occurs on rare occasions.
`accepted_for_bidding`	The order has been received by exchanges, and is evaluated for pricing. This state only occurs on rare occasions.
`stopped`	The order has been stopped, and a trade is guaranteed for the order, usually at a stated price or better, but has not yet occurred. This state only occurs on rare occasions.
`rejected`	The order has been rejected, and no further updates will occur for the order. This state occurs on rare occasions and may occur based on various conditions decided by the exchanges.
`suspended`	The order has been suspended, and is not eligible for trading. This state only occurs on rare occasions.
`calculated`	The order has been completed for the day (either filled or done for day), but remaining settlement calculations are still pending. This state only occurs on rare occasions.

An order may be canceled through the API up until the point it reaches a state of either `filled`, `canceled`, or `expired`.

Odd Lots and Block Trades

When trading stocks, a round lot is typically defined as 100 shares, or a larger number that can be evenly divided by 100. An odd lot is anything that cannot be

evenly divided by 100 shares (e.g. 48, 160, etc.). A block trade is typically defined as a trade that involves 10,000 shares or more.

For trading purposes, odd lots are typically treated like round lots. However, regulatory trading rules allow odd lots to be treated differently. Similarly, block trades are usually broken up for execution and may take longer to execute due to the market having to absorb the block of shares over time rather than in one large execution. When combined with a thinly traded stock, it's quite possible that odd lots and block trades may not get filled or execute in a timely manner, and sometimes, not at all, depending on other factors like order types used.

Short Sales

A short sale is the sale of a stock that a seller does not own. In general, a short seller sells borrowed stock in anticipation of a price decline. The short seller later closes out the position by purchasing the stock.

Order Handling Standards at Alpaca Securities LLC

Market and limit order orders are protected on the primary exchange opening print. We do not necessarily route retail orders to the exchange, but will route orders to market makers who will route orders on your behalf to the primary market opening auction. This protection is subject to exchange time cutoff for each exchange's opening process. For instance, if you enter a market order between 9:28:01 and 9:29:59 on a Nasdaq security you would not receive the Nasdaq Official Opening Price (NOOP) since Nasdaq has a cutoff of 9:28 for market orders to be sent to the cross. Any market orders received before 9:28 will be filled at the Nasdaq Official Opening Price.

Stop orders and trailing stops are elected on the consolidated print. Your sell stop order will only elect if there is a trade on the consolidated tape at or lower than your stop price and provided the electing trade is not outside of the NBBO. Your buy stop order will only elect if there is a trade on the consolidated tape that is at or above your stop price that is not outside of the NBBO.

Limit Orders are generally subject to limit order display and protection. Protection implies that you should not see the stock trade better than your limit without you receiving an execution. Limit Order Display is bound by REG NMS

Rule 611. Your orders will be displayed if they are the National Best Bid or Best Offer excluding exceptions outlined REG NMS Rule 611. Some examples are listed below:

- An odd lot order (under a unit of trade). Most NMS securities have a unit of trade of 100 shares.
- Block Order. A block order under REG NMS is designated as an order of at least 10,000 shares or at least $200,000 notional.
- An “all or none” order
- The client requests the order to not be displayed.
- Not Held orders

OTC Positions

OTC positions that are actively trading will not be automatically removed/liquidated from a user's account. The user can submit orders if the client wishes to remove or liquidate them.

OTC markets have several market tiers and depending on the market tier there may be more considerations when placing an order to close the position.

OTC Market Tier
OTCQX, OTCQB, Pink (Current, limited, No information)
Clients can enter market orders, limit orders and stop orders to close positions. For market orders, clients can mix whole shares and fractional shares.

Expert Market, Caveat Emptor
Clients must enter limit orders for whole shares, fractional shares in an order with whole shares will not be accepted. We are not able to accept a combination of a whole and fractional share for Expert Market and Caveat Emptor securities. Clients must enter a market in order to sell a fractional position.

Example: User owns 10.53 shares of WXYZ Stock: Sell Limit Order for 10 shares of WXYZ Stock and Sell Market Order for 0.53 shares of WXYZ Stock.

OTC market data is a separate data subscription. In many circumstances, a client can obtain a real-time level one quote from otcmarkets.com. Additionally, you can check the OTC market tier on that website as well.

User Protection

We have enabled several types of protections to enhance your trading experience.

1. Pattern Day Trader (PDT) Protection
2. Day Trade Margin Call (DTMC) Protection
3. Preventing Wash Trades

Please note that these do not apply to crypto trading as cryptocurrencies are not marginable. Pattern Day Trading rule does not apply to crypto trading either.

Pattern Day Trader (PDT) Protection at Alpaca

In order to prevent Alpaca Brokerage Account customers from unintentionally being designated as a Pattern Day Trader (PDT), the Alpaca Trading platform checks the PDT rule condition every time an order is submitted from a customer. If the order could potentially result in the account being flagged as a PDT, the order is rejected, and API returns error with HTTP status code 403 (Forbidden).

The Rule

A day trade is defined as a round-trip pair of trades within the same day (including extended hours). This is best described as an initial or opening transaction that is subsequently closed later in the same calendar day. For long positions, this would consist of a buy and then sell. For short positions, selling a security short and buying it back to cover the short position on the same day would also be considered a day trade.

An account is designated as a Pattern Day Trader if it makes four (4) day trades within five (5) business days. Day trades less than this criteria will not flag the account for PDT.

Cryptocurrency trading is not subject to the PDT rule. As a result, crypto orders are not evaluated by PDT protection logic and round-trip crypto trades on the same day do not contribute to the day trade count.

Day trades are counted regardless of share quantity or frequency throughout the day. Here are some FINRA-provided examples:

Example A:

09:30 Buy 250 ABC
09:31 Buy 250 ABC
13:00 Sell 500 ABC
The customer has executed one day trade.

Example B:
09:30 Buy 100 ABC
09:31 Sell 100 ABC
09:32 Buy 100 ABC
13:00 Sell 100 ABC
The customer has executed two day trades.

Example C:
09:30 Buy 500 ABC
13:00 Sell 100 ABC
13:01 Sell 100 ABC
13:03 Sell 300 ABC
The customer has executed one day trade.

Example D:
09:30 Buy 250 ABC
09:31 Buy 300 ABC
13:01 Buy 100 ABC
13:02 Sell 150 ABC
13:03 Sell 175 ABC
The customer has executed one day trade.

Example E:
09:30 Buy 199 ABC
09:31 Buy 142 ABC
13:00 Sell 1 ABC
13:01 Buy 45 ABC
13:02 Sell 100 ABC
13:03 Sell 200 ABC
The customer has executed two day trades.

Example F:
09:30 Buy 200 ABC
09:30 Buy 100 XYZ
13:00 Sell 100 ABC
13:00 Sell 100 XYZ

The customer has executed two day trades.

For further information, please visit Regulatory Notice 21-13 | FINRA.org

Alpaca's Order Rejection

Alpaca Trading platform monitors the number of day trades for the account for the past 5 business days and rejects a newly submitted orders on exit of a position if it could potentially result in the account being flagged for PDT. This protection triggers only when the previous day's closing account equity is less than $25,000 at the time of order submission.

In addition to the filled orders, the system also takes into consideration pending orders in the account. In this case, regardless of the order of pending orders, a pair of buy and sell orders is counted as a potential day trade. This is because orders that are active (pending) in the marketplace may fill in random orders. Therefore, even if your sell limit order is submitted first (without being filled yet) and another buy order on the same security is submitted later, this buy order will be blocked if your account already has 3 day trades in the last 5 business days.

Paper Trading

The same protection triggers in your paper trading account. It is advised to test your algorithm with the realistic balance amount you would manage when going live, to make sure your assumption works under this PDT protection as well.

For more details of Pattern Day Trader rule, please visit the FINRA website.

Day Trade Margin Call (DTMC) Protection at Alpaca

In order to prevent Alpaca Brokerage Account customers from unintentionally receiving day trading margin calls, Alpaca implements two forms of DTMC protection.

The Rule

Day traders are required to have a minimum of $25,000 OR 25% of the total market value of securities (whichever is higher) maintained in their account.

The buying power of a pattern day trader is 4x the excess of the maintenance margin from the closing of the previous day. If you exceed this amount, you will receive a day trading margin call.

How Alpaca's DTMC Protection Settings Work

Users only receive day trading buying power when marked as a pattern day trader. If the user is designated a pattern day trader, the account.multiplier is equal to 4.

Daytrading buying power cannot increase beyond its start of day value. In other words, closing an overnight position will not add to your daytrading buying power.

The following scenarios and protections are applicable only for accounts that are designated as pattern day traders. Please check your Account API result for the multiplier field.

Every trading day, you start with the new `daytrading_buying_power`. This beginning value is calculated as `4 * (last_equity - last_maintenance_margin)`. The last_equity and last_maintenance_margin values can be accessed through Account API. These values are stored from the end of the previous trading day.
Throughout the day, each time you enter a new position, your `daytrading_buying_power` is reduced by that amount. When you exit that position within the same day, that same amount is credited back, regardless of position's P/L.
At the end of the trading day, on close, the maximum exposure of your day trading position is checked. A Day Trade Margin Call (DTMC) is issued the next day if the maximum exposure of day trades exceeded your day trading buying power from the beginning of that day.

The buying_power value is the larger of `regt_buying_power` and `daytrading_buying_power`. Since the basic buying power check runs on this buying_power value, you could be exceeding your `daytrading_buying_power` when you enter the position if `regt_buying_power` is larger than your `daytrading_buying_power` at one point in the day.
The following is an example scenario:

1. Your equity is $50k
2. You hold overnight positions up to $100k

3. Your maintenance margin is $30k (~30%), therefore your day trading buying power at the beginning of day is $80k using the calculation of 4 * ($50k - $30k)
4. You sell all of the overnight positions ($100k value) in the morning, which brings your `regt_buying_power` up to $100k
5. You now buy and sell the same security up to $100k
6. At the end of the day, you have a $20k Day Trade Margin Call ($100k - $80k)

By default, Alpaca users have DTMC protections on entry of a position. This means that if your entering order would exceed `daytrading_buying_power` at the moment, it will be blocked, even if `regt_buying_power` still has room for it. This is based on the assumption that any entering position could be day trades later in the day. This option is the more conservative of the two DTMC protections that our users have.
The second DTMC protection option is protection on exit of a position. This means that Alpaca will block the exit of positions that would cause a Day Trading Margin Call. This may cause users to be unable to liquidate a position until the next day.

Neither of the DTMC protection options evaluate crypto orders since crypto cannot be purchased using margin.

One of the two protections will be enabled for all users (you cannot have both protections disabled). If you would like to switch your protection option, please contact our support.

We are working towards features to allow users to change their DTMC protection setting on their own without support help.

Equity/Order Ratio Validation Check

In order to help Alpaca Brokerage Account customers from placing orders larger than the calculated buying power, Alpaca has instituted a control on the account independent of the buying power for the account. Alpaca will restrict the account to closing transactions when an account has a position that is 600% larger than the equity in the account. The account will remain restricted for closing transactions until a member of Alpaca's trading team reviews the account. The trading team will either clear the alert by allowing opening transactions or will notify the client of the restriction and take corrective actions as necessary.

Paper Trading

he same protection triggers in your paper trading account. It is advised to test your algorithm with the realistic balance amount you would manage when going live, to make sure your assumption works under this DTMC protection as well.

For more details of Pattern Day Trader rule, please read FINRA's margin requirements. For more details on day trade margins, please read FINRA's Mind Your Margin article.

Preventing Wash Trades at Alpaca

At Alpaca, we want to help our customers avoid making unintentional wash trades. A wash trade happens when a customer buys and sells the same security at the same time, which can be seen as a form of market manipulation. To prevent this, the Alpaca Trading platform checks for potential wash trades every time a customer places an order. If we detect a possible wash trade, we reject the order and send back an error message with the HTTP status code 403 (Forbidden).

The Rule

A wash trade occurs when a customer's two orders could potentially interact with each other. Here are a couple of examples:

- A customer places an order to buy 1 share at $10 (a limit order). Then, the same customer places an order to sell 100 shares at $10 (another limit order). These orders could potentially interact, which would be a wash trade.
- A customer places an order to sell 100 shares at the market open (a market order). Then, the same customer places an order to buy 100 shares at $10 (a limit order). Again, these orders could potentially interact, which would be a wash trade.

How Alpaca Handles Potential Wash Trades

The Alpaca Trading platform is always on the lookout for potential wash trades. If we determine that an order could result in a wash trade, we trigger our

protection measures, reject the order, and send back an error message with the HTTP status code 403 (Forbidden).

If a customer wants to set up a 'take profit' and a 'stop loss' situation, we recommend using a bracket or OCO (One Cancels the Other) order. These complex orders and trailing stop orders are exceptions to our wash trade protection.

Here's a table that shows when we would reject an order to prevent a potential wash trade:

Existing Order	New Order	Reject Condition
market buy	market sell	always rejected
market buy	limit sell	always rejected
market buy	stop sell	always rejected
market buy	stop_limit sell	always rejected
market sell	market buy	always rejected
market sell	limit buy	always rejected
market sell	stop buy	always rejected
market sell	stop_limit buy	always rejected
stop buy	market sell	always rejected
stop buy	limit sell	always rejected
stop buy	stop sell	always rejected
stop buy	stop_limit sell	always rejected
stop sell	market buy	always rejected
stop sell	limit buy	always rejected
stop sell	stop buy	always rejected
stop sell	stop_limit buy	always rejected
limit buy	market sell	always rejected
limit buy	limit sell	rejected if buy limit price >= sell limit price
limit buy	stop sell	always rejected
limit buy	stop_limit sell	rejected if buy limit price >= sell limit price

Existing Order	New Order	Reject Condition
limit sell	market buy	always rejected
limit sell	limit buy	rejected if buy limit price >= sell limit price
limit sell	stop buy	always rejected
limit sell	stop_limit buy	rejected if buy limit price >= sell limit price
stop_limit buy	market sell	always rejected
stop_limit buy	limit sell	rejected if buy limit price >= sell limit price
stop_limit buy	stop sell	always rejected
stop_limit buy	stop_limit sell	rejected if buy limit price >= sell limit price
stop_limit sell	market buy	always rejected
stop_limit sell	limit buy	rejected if buy limit price >= sell limit price
stop_limit sell	stop buy	always rejected
stop_limit sell	stop_limit buy	rejected if buy limit price >= sell limit price

Paper Trading

Our wash trade protection also applies to your paper trading account. We recommend testing your trading algorithm with a realistic balance amount. This way, you can make sure your strategy works under our wash trade protection rules before you start live trading.

For more details of wash trade rule, please read FINRA's self-trades requirements.

Websocket Streaming

Learn how to stream market data using Websockets.

Suggest Edits

Alpaca's API offers WebSocket streaming for trade, account, and order updates which follows the RFC6455 WebSocket protocol.

To connect to the WebSocket follow the standard opening handshake as defined by the RFC specification to `wss://paper-api.alpaca.markets/stream` or `wss://api.alpaca.markets/stream`. Alpaca's streaming service supports both JSON and MessagePack codecs.
Once the connection is authorized, the client can listen to the `trade_updates` stream to get updates on trade, account, and order changes.

Note:

The `trade_updates` stream coming from `wss://paper-api.alpaca.markets/stream` uses binary frames, which differs from the text frames that comes from the `wss://data.alpaca.markets/stream` stream.

In order to listen to streams, the client sends a `listen` message to the server as follows:

JSON

```
{
  "action": "listen",
  "data": {
    "streams": ["trade_updates"]
  }
}
```

The server acknowledges by replying a message in the listening stream.

JSON

```
{
  "stream": "listening",
  "data": {
    "streams": ["trade_updates"]
  }
}
```

If any of the requested streams are not available, they will not appear in the streams list in the acknowledgement. Note that the streams field in the listen message is to tell the set of streams to listen, so if you want to stop receiving updates from the stream, you must send an empty list of streams values as a listen message. Similarly, if you want to add more streams to get updates in addition to the ones you are already doing so, you must send all the stream names, not only the new ones.

Subscribing to real-time trade updates ensures that a user always has the most up-to-date picture of their account actvivity.

Note

To request with MessagePack, add the header: Content-Type: `application/msgpack`.

Authentication

The WebSocket client can be authenticated using the same API key when making HTTP requests. Upon connecting to the WebSocket, client must send an authentication message over the WebSocket connection with the API key and secret key as its payload.

JSON
```
{
  "action": "auth",
  "key": "{YOUR_API_KEY_ID}",
  "secret": "{YOUR_API_SECRET_KEY}"
}
```

The server will then authorize the connection and respond with either an authorized (successful) response

JSON
```
{
  "stream": "authorization",
  "data": {
    "status": "authorized",
    "action": "authenticate"
  }
}
```

or an unauthorized (unsuccessful) response:

JSON
```
{
  "stream": "authorization",
  "data": {
    "status": "unauthorized",
    "action": "authenticate"
  }
}
```

In the case that the socket connection is not authorized yet, a new message under the authorization stream is issued in response to the listen request.

JSON
```
{
  "stream": "authorization",
  "data": {
    "status": "unauthorized",
    "action": "listen"
  }
}
```

Trade Updates

With regards to the account associated with the authorized API keys, updates for orders placed at Alpaca are dispatched over the WebSocket connection under the

event trade_updates. These messages include any data pertaining to orders that are executed with Alpaca. This includes order fills, partial fills, cancellations and rejections of orders. Clients may listen to this stream by sending a listen message:

JSON
```
{
  "action": "listen",
  "data": {
    "streams": ["trade_updates"]
  }
}
```

Any listen messages received by the server will be acknowledged via a message on the listening stream. The message's data payload will include the list of streams the client is currently listening to:

JSON
```
{
  "stream": "listening",
  "data": {
    "streams": ["trade_updates"]
  }
}
```

The fields present in a message sent over the `trade_updates` stream depend on the type of event they are communicating. All messages contain an `event` type and an `order` field, which is the same as the order object that is returned from the REST API. Potential event types and additional fields that will be in their messages are listed below.

Common Events

These are the events that are the expected results of actions you may have taken by sending API requests.

- `new`: Sent when an order has been routed to exchanges for execution.
- `fill`: Sent when an order has been completely filled.
  - `timestamp`: The time at which the order was filled.
  - `price`: The average price per share at which the order was filled.
  - `position_qty`: The total size of your position after this event, in shares. Positive for long positions, negative for short positions.
- `partial_fill`: Sent when a number of shares less than the total remaining quantity on your order has been filled.
  - `timestamp`: The time at which the order was partially filled.
  - `price`: The average price per share at which the order was filled.

    - `position_qty`: The total size of your position after this event, in shares. Positive for long positions, negative for short positions.
- `canceled`: Sent when your requested cancelation of an order is processed.
    - `timestamp`: The time at which the order was canceled.
- `expired`: Sent when an order has reached the end of its lifespan, as determined by the order's time in force value.
    - `timestamp`: The time at which the order was expired.
- `done_for_day`: Sent when the order is done executing for the day, and will not receive further updates until the next trading day.
- `replaced`: Sent when your requested replacement of an order is processed.
    - `timestamp`: The time at which the order was replaced.

Less Common Events

These are events that may rarely be sent due to uncommon circumstances on the exchanges. It is unlikely you will need to design your code around them, but you may still wish to account for the possibility that they can occur.

- `rejected`: Sent when your order has been rejected.
    - `timestamp`: The time at which the order was rejected.
- `pending_new`: Sent when the order has been received by Alpaca and routed to the exchanges, but has not yet been accepted for execution.
- `stopped`: Sent when your order has been stopped, and a trade is guaranteed for the order, usually at a stated price or better, but has not yet occurred.
- `pending_cancel`: Sent when the order is awaiting cancelation. Most cancelations will occur without the order entering this state.
- `pending_replace`: Sent when the order is awaiting replacement.
- `calculated`: Sent when the order has been completed for the day - it is either "filled" or "done_for_day" - but remaining settlement calculations are still pending.
- `suspended`: Sent when the order has been suspended and is not eligible for trading.
- `order_replace_rejected`: Sent when the order replace has been rejected.
- `order_cancel_rejected`: Sent when the order cancel has been rejected.

Example

An example of a message sent over the `trade_updates` stream looks like:
JSON


```
{
  "stream": "trade_updates",
  "data": {
    "event": "fill",
    "execution_id": "2f63ea93-423d-4169-b3f6-3fdafc10c418",
    "order": {
      "asset_class": "crypto",
      "asset_id": "1cf35270-99ee-44e2-a77f-6fab902c7f80",
      "cancel_requested_at": null,
      "canceled_at": null,
      "client_order_id": "4642fd68-d59a-47d7-a9ac-e22f536828d1",
      "created_at": "2022-04-19T13:45:04.981350886-04:00",
      "expired_at": null,
      "extended_hours": false,
      "failed_at": null,
      "filled_at": "2022-04-19T17:45:05.024916716Z",
      "filled_avg_price": "105.8988475",
      "filled_qty": "1790.86",
      "hwm": null,
      "id": "a5be8f5e-fdfa-41f5-a644-7a74fe947a8f",
      "legs": null,
      "limit_price": null,
      "notional": null,
      "order_class": "",
      "order_type": "market",
      "qty": "1790.86",
      "replaced_at": null,
      "replaced_by": null,
      "replaces": null,
      "side": "sell",
      "status": "filled",
      "stop_price": null,
      "submitted_at": "2022-04-19T13:45:04.980944666-04:00",
      "symbol": "SOLUSD",
      "time_in_force": "gtc",
      "trail_percent": null,
      "trail_price": null,
      "type": "market",
```


Position Average Entry Price Calculation

How is the average entry price of a position is calculated?

Suggest Edits

Description

The average entry price and the cost basis of a position are returned in the `avg_entry_price` and `cost_basis` fields in the positions endpoints.
JSON


```
{
  "asset_id": "904837e3-3b76-47ec-b432-046db621571b",
  "symbol": "AAPL ",
  "exchange": "NASDAQ",
  "asset_class": "us_equity",
```

```
  "avg_entry_price": "100.0",
  "qty": "5",
  "qty_available": "4",
  "side": "long",
  "market_value": "600.0",
  "cost_basis": "500.0",
  "unrealized_pl": "100.0",
  "unrealized_plpc": "0.20",
  "unrealized_intraday_pl": "10.0",
  "unrealized_intraday_plpc": "0.0084",
  "current_price": "120.0",
  "lastday_price": "119.0",
  "change_today": "0.0084"
}
```


There are different methods that can be used to calculate the cost basis and the average entry price of a position such as `Strict FIFO`, `Compressed FIFO`, `Weighted Average`, and others. Each method has its own rules for calculating the cost basis and average entry price after a sell transaction. This page aims to clarify which method is Alpaca using.

Which Method is Alpaca Using?

- Weighted Average is used for intraday positions (positions from intraday trades)
- Compressed FIFO is used for the end-of-day positions (positions from previous trading days)

Strict FIFO (First-In, First-Out)

Under the Strict FIFO method, the first position bought is the first position sold. Let's understand how it works:

The cost basis after the sell is calculated by deducting from the previous cost basis the price of the first open position multiplied by the sell quantity. In Strict FIFO, the sell quantity is covered using the first open position, however, if the first open position's quantity is not enough to cover the sell quantity, subsequent open positions are used.

Example:

Suppose we have the following transactions:

Day 1:

1. Buy 100 shares at $10 per share (Cost basis = $1,000)

2. Buy 50 shares at $12 per share (Cost basis = $600)

Day 2:

1. Buy 30 shares at $15 per share (Cost basis = $450)

Day 3:

1. Sell 120 shares

After the sell transaction:

- Cost basis: `2050 - 100*10 - 20*12` = `$810`
- Average Entry Price: `cost_basis/qty_left` = `810/60` = `$13.5`

Compressed FIFO (First-In, First-Out)

The Compressed FIFO method follows similar rules to Strict FIFO, with one key difference. It compresses intraday positions using a weighted average. Let's see how it differs:

Example 1:

Using the same example from before:
Day 1:

1. Buy 100 shares at $10 per share (Cost basis = $1,000)
2. Buy 50 shares at $12 per share (Cost basis = $600)

Day 2:

1. Buy 30 shares at $15 per share (Cost basis = $450)

Day 3:

1. Sell 120 shares

After the sell transaction:

- Cost Basis: `2050 - 120*(100*10 + 50*12)/150` = `$770`
- Average entry price: `cost_basis/qty_left` = `770/60` = `$12.83`

As you can see the positions in Day 1 were compressed into a total of 150 shares with an average price of `(100*10 + 50*12)/150`.

Example 2

Day 1:

1. Buy 100 shares at $10 per share (Cost basis = $1,000)
2. Buy 50 shares at $9 per share (Cost basis = $450)
3. Sell 50 shares
4. Buy 50 shares at $11 per share (Cost basis = $550)

At the end of Day 1:

- Cost Basis: `2000 - 50*(100*10 + 50*9 + 50*11)/200` = `$1,500`
- Average Entry Price: `1500/150` = `$10`

Weighted Average

The Weighted Average method calculates the cost basis based on the weighted average price per share. Here's how it works:

On Sell: The cost basis for the sold quantity is calculated by deducting the sell quantity multiplied by the average entry price of all the opened positions that the account holds.

Example:

Using the same example from before:

Day 1:

1. Buy 100 shares at $10 per share (Cost basis = $1,000)
2. Buy 50 shares at $12 per share (Cost basis = $600)

Day 2:

1. Buy 30 shares at $15 per share (Cost basis = $450)

Day 3:

1. Sell 120 shares

After the sell the calculations based on the Weighted average method would be:

- Cost Basis: `2050 - 120*(100*10 + 50*12 + 30*15)/180` = `$683.33`
- Average Entry Price: `cost_basis/qty_left` = `683.33/60` = `$11.39`

FAQ

Why did the `avg_entry_price` and `cost_basis` of a position change the next day?

As described in the Which method is Alpaca using? the calculation method for determining the `avg_entry_price` and `cost_basis` differs between the `intraday positions` and the `end-of-day positions`. Consequently, it is possible for the `avg_entry_price` and `cost_basis` fields of a position to change the day after the last trade has occurred. This change occurs when our beginning-of-day (BOD) job executes and synchronizes positions from our ledger. For details regarding the timing of the beginning-of-day (BOD) job, please refer to the Daily Processes and Reconciliations.

About Market Data API

Gain seamless access to a wealth of data with Alpaca Market Data API, offering real-time and historical equities & crypto information.

Suggest Edits

Overview

The Market Data API v2 offers seamless access to market data through both HTTP and WebSocket protocols. With a focus on historical and real-time data, developers can efficiently integrate these APIs into their applications.

To simplify the integration process, we provide user-friendly SDKs in Python, Go, NodeJS, and C#. These SDKs offer comprehensive functionalities, making it easier for developers to work with the Market Data APIs & Web Sockets.

To start using the APIs, developers have two convenient options: they can either access it through the public workspace on Postman or directly from our GitHub repository.

By leveraging Alpaca Market Data API v2 and its associated SDKs, developers can seamlessly incorporate historical and real-time market data into their applications, enabling them to build powerful and data-driven financial products.

Subscription Plans

The Market Data API v2 offers Market Data access under three distinct plans: **Free**, **Algo Trader Plus**, and **Broker Professional**.

The Free plan serves as the default option for both Paper and Live trading accounts, ensuring all users can access essential data with zero cost.

In addition to the Free plan, we also provide two premium options: Algo Trader Plus and Broker Professional. These plans cater to the needs of traders and brokers who require advanced data features and increased data usage allowances.

Equities

	Free	**Algo Trader Plus**	**Broker Professional**
Pricing	Free	$99/month	$99/mo/device
Securities coverage	US Stocks & ETFs	US Stocks & ETFs	US Stocks & ETFs
Real-time market coverage	IEX	All US Stock Exchanges	All US Stock Exchanges
Websocket subscriptions	30 symbols	Unlimited	Unlimited
Historical data timeframe	Since 2016	Since 2016	Since 2016
Historical data limitation*	latest 15 minutes	no restriction	no restriction
Historical API calls	200/min	10,000/min	10,000/min

Our data sources are directly fed by the CTA (Consolidated Tape Association), which is administered by NYSE (New York Stock Exchange), and the UTP (Unlisted Trading Privileges) stream, which is administered by Nasdaq. The synergy of these two sources ensures comprehensive market coverage, encompassing 100% of market volume.

Options

	Free	Algo Trader Plus
Securities coverage	US Options Securities	US Options Securities
Real-time market coverage	Indicative Pricing Feed	OPRA Feed
Websocket subscriptions	200 quotes	1000 quotes
Historical data limitation*	latest 15 minutes	no restriction
Historical API calls	200/min	10,000/min

Our options data sources are directly fed by OPRA (Options Price Reporting Authority).

Free vs Algo Trader Plus*

1. Users on the Free Market Data plan have access to the IEX (Investors Exchange LLC) stream but with some limitations. Historical equities data is available without restrictions, however, it is not allowed to query the latest 15-minute data.
2. This also means that users on the Free Market Data plan may not be able to access the most recent SIP (Securities Information Processor) data or use certain latest endpoints that rely exclusively on SIP data.
3. Moreover, users on the Free Market Data plan can access Alpaca's Indicative Pricing Feed for free. The same limitation applies to historical options data as equities (IEX), users can only query more than 15-minute old data.

Broker Professional

We offer custom pricing and tailored solutions for Broker API partners seeking to leverage our comprehensive market data. Our goal is to meet the specific needs and requirements of our valued partners, ensuring they have access to the data and tools necessary to enhance their services and provide exceptional value to their customers.

For detailed information about our pricing options and the benefits of becoming a Broker API partner, kindly reach out to our sales team.

Exchanges

Alpaca supports a range of stock exchanges, each identified by a unique tape ID, which is returned in all market data requests. To help you easily map the tape code to the corresponding exchange, here is the list:

Exchange Code	Exchange Name
A	NYSE American (AMEX)
B	NASDAQ OMX BX
C	National Stock Exchange
D	FINRA ADF
E	Market Independent
H	MIAX
I	International Securities Exchange
J	Cboe EDGA
K	Cboe EDGX
L	Long Term Stock Exchange
M	Chicago Stock Exchange
N	New York Stock Exchange
P	NYSE Arca
Q	NASDAQ OMX
S	NASDAQ Small Cap
T	NASDAQ Int
U	Members Exchange
V	IEX
W	CBOE
X	NASDAQ OMX PSX
Y	Cboe BYX
Z	Cboe BZX

Conditions

Every feed or stock exchange utilizes its unique set of codes to identify trade and quote conditions. Consequently, the same condition may vary in code representation depending on the data's originator.

Trade conditions

CTS

Below is a table containing codes that indicate specific trade conditions applicable under the CTA (Consolidated Tape Association) Plan:

For further details on this topic, please refer to page 64 of the Consolidated Tape System (CTS) Specification.

Code	Value
`Space`	Regular Sale
B	Average Price Trade
C	Cash Trade (Same Day Clearing)
E	Automatic Execution
F	Inter-market Sweep Order
H	Price Variation Trade
I	Odd Lot Trade
K	Rule 127 (NYSE only) or Rule 155 (NYSE MKT only)
L	Sold Last (Late Reporting)
M	Market Center Official Close
N	Next Day Trade (Next Day Clearing)
O	Market Center Opening Trade
P	Prior Reference Price
Q	Market Center Official Open
R	Seller
T	Extended Hours Trade
U	Extended Hours Sold (Out Of Sequence)
V	Contingent Trade

Code	Value
X	Cross Trade
Z	Sold (Out Of Sequence)
4	Derivatively Priced
5	Market Center Reopening Trade
6	Market Center Closing Trade
7	Qualified Contingent Trade
8	Reserved
9	Corrected Consolidated Close Price as per Listing Market

UTDF

Below is a table containing condition codes from the UTP (Unlisted Trading Privileges) Plan:

For further details on this topic, please refer to page 43 of the UTP Specification.

Code	Value
@	Regular Sale
A	Acquisition
B	Bunched Trade
C	Cash Sale
D	Distribution
E	Placeholder
F	Intermarket Sweep
G	Bunched Sold Trade
H	Price Variation Trade
I	Odd Lot Trade`
K	Rule 155 Trade (AMEX)
L	Sold Last
M	Market Center Official Close

Code	Value
N	Next Day
O	Opening Prints
P	Prior Reference Price
Q	Market Center Official Open
R	Seller
S	Split Trade
T	Form T
U	Extended trading hours (Sold Out of Sequence)
V	Contingent Trade
W	Average Price Trade
X	Cross Trade
Y	Yellow Flag Regular Trade
Z	Sold (Out Of Sequence)
1	Stopped Stock (Regular Trade)
4	Derivatively Priced
5	Re-Opening Prints
6	Closing Prints
7	Qualified Contingent Trade (QCT)
8	Placeholder For 611 Exempt
9	Corrected Consolidated Close (per listing market)

Quote conditions

CQS

Below is a table containing codes that indicate specific conditions applicable to a quote under the CTA (Consolidated Tape Association) Plan:

For further details on this topic, please refer to Appendix G of the CQS Specification.

Code	Value
A	Slow Quote Offer Side
B	Slow Quote Bid Side
E	Slow Quote LRP Bid Side
F	Slow Quote LRP Offer Side
H	Slow Quote Bid And Offer Side
O	Opening Quote
R	Regular Market Maker Open
W	Slow Quote Set Slow List
C	Closing Quote
L	Market Maker Quotes Closed
U	Slow Quote LRP Bid And Offer
N	Non Firm Quote
4	On Demand Intra Day Auction

UQDF

Below is a table containing codes that represent specific conditions applicable to quotes under the UTP (Unlisted Trading Privileges) Plan:

For further details on this topic, please refer to the UQDF Specification.

Code	Value
A	Manual Ask Automated Bid
B	Manual Bid Automated Ask
F	Fast Trading
H	Manual Bid And Ask
I	Order Imbalance
L	Closed Quote
N	Non Firm Quote
O	Opening Quote Automated

Code	Value
R	Regular Two Sided Open
U	Manual Bid And Ask Non Firm
Y	No Offer No Bid One Sided Open
X	Order Influx
Z	No Open No Resume
4	On Demand Intra Day Auction

Getting Started with Market Data API

This is a quick guide on how to start consuming market data via APIs. Starting from beginning to end, this section outlines how to install Alpaca's software development kit (SDK), create a free alpaca account, locate your API keys, and how to request both historical and real-time data.

Suggest Edits

Installing Alpaca's Client SDK

In this guide, we'll be making use of the SDKs provided by Alpaca. Alpaca maintains SDKs in four languages: Python, JavaScript, C#, and Go. Follow the steps in the installation guide below to install the SDK of your choice before proceeding to the next section.

PythonJavaScriptC#Go

```
pip install alpaca-py
```

Generate API Keys

How to Request Market Data Through the SDK

With the SDK installed and our API keys ready, you can start requesting market data. Alpaca offers many options for both historical and real-time data, so to keep this guide succint, these examples are on obtaining historical and real-time bar data. Information on what other data is available can be found in the Market Data API reference.

To start using the SDK for historical data, import the SDK and instantiate the crypto historical data client. It's not required for this client to pass in API keys or a paper URL.

Python Go

```python
from alpaca.data.historical import CryptoHistoricalDataClient

# No keys required for crypto data
client = CryptoHistoricalDataClient()
```

Next we'll define the parameters for our request. Import the request class for crypto bars, CryptoBarsRequest and TimeFrame class to access time frame units more easily. This example queries for historical daily bar data of Bitcoin in the first week of September 2022.

Python Go

```python
from alpaca.data.requests import CryptoBarsRequest
from alpaca.data.timeframe import TimeFrame

# Creating request object
request_params = CryptoBarsRequest(
  symbol_or_symbols=["BTC/USD"],
  timeframe=TimeFrame.Day,
  start="2022-09-01",
  end="2022-09-07"
)
```

Finally, send the request using the client's built-in method, get_crypto_bars. Additionally, we'll access the .df property which returns a pandas DataFrame of the response.

Python Go

```python
# Retrieve daily bars for Bitcoin in a DataFrame and printing it
btc_bars = client.get_crypto_bars(request_params)

# Convert to dataframe
btc_bars.df
```

Returns

Text

```
                           open    high    low     close   volume  trade_count     vwap
symbol  timestamp
BTC/USD
        2022-09-01 05:00:00+00:00       20049.0 20285.0 19555.0 20160.0 2396.3504
        18060.0 19920.278135
        2022-09-02 05:00:00+00:00       20159.0 20438.0 19746.0 19924.0 1688.0641
        16730.0 20045.987764
        2022-09-03 05:00:00+00:00       19924.0 19963.0 19661.0 19802.0 624.1013
        9853.0  19794.111057
        2022-09-04 05:00:00+00:00       19801.0 20060.0 19599.0 19892.0 1361.6668
        8489.0  19885.445568
```

```
2022-09-05 05:00:00+00:00       19892.0 20173.0 19640.0 19762.0 2105.0539
11900.0 19814.853546
2022-09-06 05:00:00+00:00       19763.0 20025.0 18539.0 18720.0 3291.1657
19591.0 19272.505607
2022-09-07 05:00:00+00:00       18723.0 19459.0 18678.0 19351.0 2259.2351
16204.0 19123.487500
```

Request ID

All market data API endpoint provides a unique identifier of the API call in the response header with `X-Request-ID` key, the Request ID helps us to identify the call chain in our system.
Make sure you provide the Request ID in all support requests that you created, it could help us to solve the issue as soon as possible. Request ID can't be queried in other endpoints, that is why we suggest to persist the recent Request IDs.

Shell

```
$ curl -v https://data.alpaca.markets/v2/stocks/bars
...
> GET /v2/stocks/bars HTTP/1.1
> Host: data.alpaca.markets
> User-Agent: curl/7.88.1
> Accept: */*
>
< HTTP/1.1 403 Forbidden
< Date: Fri, 25 Aug 2023 09:37:03 GMT
< Content-Type: application/json
< Content-Length: 26
< Connection: keep-alive
< X-Request-ID: 0d29ba8d9a51ee0eb4e7bbaa9acff223
<
...
```

Stock Pricing API

The Stock Pricing API offered by Alpaca Markets provides developers with access to real-time and historical pricing data for various stocks.
Suggest Edits
This powerful API allows users to integrate stock market information into their applications, enabling them to build innovative financial tools and strategies.

Data Sources

Alpaca offers market data from two distinct data sources:

Source	Description
IEX (Investors Exchange LLC)	IEX is ideal for initial app testing and situations where precise price visualization may n primary focus. It accounts for approximately ~2.5% of the market volume. This Alpaca data feed originates directly from exchanges and is consolidated by the Secu Information Processors (SIPs). These SIPs play a crucial role in connecting various U.S. processing and consolidating all bid/ask quotes and trades from multiple trading venues single, easily accessible data feed.
SIP (All US Exchanges)	Our data delivery ensures ultra-low latency and high reliability, as the information is tra directly to Alpaca's bare metal servers located in New Jersey, situated alongside many m participants. SIP data is particularly advantageous for developing your trading app, where precise and date price information is essential for traders and internal operations. It accounts for 100 market volume, providing comprehensive coverage for your trading needs.

Environments

Trading API

Base URL

- `https://data.alpaca.markets/v2`

URL

- `wss://stream.data.alpaca.markets/v2/{source}` *where source is either* ***iex*** *or* ***sip****.*

Broker API

Broker API Sandbox Environment

With your Broker API key, you can freely access the IEX market data. This data access is completely cost-free and allows you to redistribute it to your end users without any additional agreements or charges.

Base URL

- `https://data.sandbox.alpaca.markets/v2`

URL

- `wss://stream.data.sandbox.alpaca.markets/v2/{source}` *where source is either* ***iex*** *or* ***sip****.*

Broker API Limited Live & Live Environment

To access Alpaca's SIP data, additional agreements with corresponding exchanges and associated charges are involved. If you require this integration for your Limited Live & Live environment, kindly reach out to our sales team.

News API

Alpaca, in partnership with Benzinga, offers historical news data through a REST API and real-time access through WebSockets.

Suggest Edits

Use Cases

News API is a versatile tool that can be used to support a variety of use cases, such as building an app with the Broker API or Algorithmic Trading using Sentiment Analysis on News with the Trading API.

1. **News Widgets**

   News API can be used to create visual news widgets for web and mobile apps. These widgets can be used to display the latest news for any stock or crypto symbol, and they include different sized images to give your app a visual appeal.

2. **News Sentiment Analysis**
   News API can be used to train models that can determine the sentiment of a given headline or news content. This can be done by using historical data from News API to train the model on a variety of different sentiment labels.

3. **Realtime Trading on News**
   Real-time news over WebSockets can be used to enable your trading algorithms to react to the latest news across any stock or cryptocurrency. This can be done by subscribing to a WebSockets feed from a news provider, such as News API.

Historical (REST)

News API provides historical news data dating back to 2015. You can expect to receive an average of 130+ news articles per day. All news data is currently provided directly by Benzinga.

With a single endpoint, you can request news for both stocks and cryptocurrency tickers. For a more comprehensive reference of News API, please see the following API reference:

Get Latest News

By calling the News API with no query parameters, you will get the latest 10 news articles from all sources. This means that you will get a mix of news articles about stocks, cryptocurrencies, and other topics.

cURL

```
curl --location --request GET 'https://data.alpaca.markets/v1beta1/news' \
--header 'Apca-Api-Key-Id: <KEY>' \
--header 'Apca-Api-Secret-Key: <SECRET>'
```

Here is an example of the response you will receive when you call the News API with no query parameters:

JSON


```
{
    "news": [
        {
            "id": 24919570,
            "headline": "Rumble Announces 'Exclusive' Content From AMC Ape Matt
Kohrs And Rand Paul, Record Site Numbers Revealed",
            "author": "Chris Katje",
            "created_at": "2022-01-05T22:51:39Z",
            "updated_at": "2022-01-05T23:31:19Z",
            "summary": "Video platform Rumble shared several company updates
Wednesday, including new user metrics.",
            "url": "https://www.benzinga.com/news/22/01/24919570/rumble-announces-
amc-ape-matt-kohrs-and-rand-paul-joining-record-site-numbers-revealed",
            "images": [
                {
                    "size": "large",
                    "url":
"https://cdn.benzinga.com/files/imagecache/2048x1536xUP/images/story/2012/mattkohr
sonrumble.png"
                },
                {
                    "size": "small",
                    "url":
"https://cdn.benzinga.com/files/imagecache/1024x768xUP/images/story/2012/mattkohrs
onrumble.png"
                },
                {
                    "size": "thumb",
```

```
                    "url":
"https://cdn.benzinga.com/files/imagecache/250x187xUP/images/story/2012/mattkohrso
nrumble.png"
                }
            ],
            "symbols": [
                "AMC",
                "AMZN",
                "CFVI",
                "GME",
                "GOOG",
                "GOOGL",
                "TWTR"
            ],
            "source": "benzinga"
        }
    ],
    "next_page_token": "MTY0MTQyMjU0OTAwMDAwMDAwMHwyNDkxNzQzNw=="
}
```


Get News for Multiple Symbols

To get news articles for multiple stocks or cryptocurrencies, you can provide a comma-separated list of symbols. For example, the following request will get news articles for both Coinbase (COIN) and bitcoin (BTCUSD):

cURL

```
curl --location --request GET
'https://data.alpaca.markets/v1beta1/news?symbols=COIN,BTCUSD' \
--header 'Apca-Api-Key-Id: <KEY>' \
--header 'Apca-Api-Secret-Key: <SECRET>'
```

JSON


```
{
  "id": 24899683,
  "headline": "Investment Firm Offers Diversified Exposure To Crypto Innovations:
Bitcoin, Solana, Avalanche And More",
  "author": "Renato Capelj",
  "created_at": "2022-01-05T20:11:10Z",
  "updated_at": "2022-01-05T20:23:40Z",
  "summary": "It takes time for innovations to penetrate their target markets. The
rate of adoption is best characterized by the s-curve, a mathematical graph that
plots growth against time.\n\nAt the outset, growth is slow in relation to the
time that passes. With time, however, growth accelerates rapidly as the majority
adopt the innovation.",
  "url": "https://www.benzinga.com/fintech/22/01/24899683/investment-firm-offers-
diversified-exposure-to-crypto-innovations-bitcoin-solana-avalanche-and-more",
   "images": [
      {
         "size": "large",
         "url":
"https://cdn.benzinga.com/files/imagecache/2048x1536xUP/images/story/2012/cover_ph
oto_66.png"
      },
      {
         "size": "small",
```

```
          "url":
"https://cdn.benzinga.com/files/imagecache/1024x768xUP/images/story/2012/cover_pho
to_66.png"
        },
        {
            "size": "thumb",
            "url":
"https://cdn.benzinga.com/files/imagecache/250x187xUP/images/story/2012/cover_phot
o_66.png"
        }
    ],
    "symbols": [
        "AVAXUSD",
        "BTCUSD",
        "BX",
        "COIN",
        "ETHUSD",
        "SOLUSD"
     ],
     "source": "benzinga"
}
```


Real-time (STREAMING)

Real-time news streaming over WebSocket delivers between 600 and 900 news articles or headlines per day.

Real-time News

You can use the `wscat` or `websocat` command-line tools to test a connection to the news stream.
In the example below, you can see how to authenticate with the server and subscribe to news for any symbol.

```
wscat -c wss://stream.data.alpaca.markets/v1beta1/news
Connected (press CTRL+C to quit)
< [{"T":"success","msg":"connected"}]

> {"action":"auth","key":"<KEY>","secret":"<SECRET>"}
< [{"T":"success","msg":"authenticated"}]

> {"action":"subscribe","news":["*"]}
< [{"T":"subscription","news":["*"]}]

<[{"T":"n","id":24919710,"headline":"Granite Wins $90M Construction
Manager/General Contractor Project In Northern California","summary":"Granite
(NYSE:GVA) announced today that it has been selected by the California Department
of Transportation (Caltrans) as the Construction Manager/General Contractor
(CM/GC) for the approximately $90 million State Route","author":"Benzinga
Newsdesk","created_at":"2022-01-05T22:30:29Z","updated_at":"2022-01-
05T22:30:30Z","url":"https://www.benzinga.com/news/22/01/24919710/granite-wins-
90m-construction-managergeneral-contractor-project-in-northern-
```

california","content":"\u003cp\u003eGranite
(NYSE:...","symbols":["GVA"],"source":"benzinga"}]

WebSocket Stream

Suggest Edits

This API provides a WebSocket stream for real-time market data. This allows you to receive the most up-to-date market information, which can be used to power your trading strategies.

The WebSocket stream provides real-time updates of the following market data:

- Stocks
- Crypto
- Options
- News

Steps to use the stream

To use the WebSocket stream follow these steps:

Connection

To establish a connection use the stream URL depending on the data you'd like to consume. The general schema of the URL is

```
wss://stream.data.alpaca.markets/{version}/{feed}
```

Sandbox URL:

```
wss://stream.data.sandbox.alpaca.markets/{version}/{feed}
```

Any attempt to access a data feed not available for your subscription will result in an error during authentication.

We provide a test stream that is available all the time, even outside market hours, on this URL:

```
wss://stream.data.alpaca.markets/v2/test
```

Upon successfully connecting, you will receive the welcome message:

JSON

```
[{"T":"success","msg":"connected"}]
```

Connection limit

The number of connections to a single endpoint from a user is limited based on the user's subscription, but in most subscriptions (including Algo Trader Plus) this limit is 1. If you try to open a second connection, you'll get this error:

JSON
```
[{"T":"error","code":406,"msg":"connection limit exceeded"}]
```

Authentication

You need to authenticate yourself using your credentials. This can be done two ways

Authenticate with HTTP headers

You can set the same headers used for the historical market data and trading endpoints:

- `APCA-API-KEY-ID`
- `APCA-API-SECRET-KEY`

Here's an example using a WebSocket client called websocat:

```
$ websocat wss://stream.data.alpaca.markets/v2/test \
  -H="APCA-API-KEY-ID: {KEY_ID}" -H="APCA-API-SECRET-KEY: {SECRET}"
```

Authenticate with a message

Alternatively, you can authenticate with a message after connection:

JSON
```
{"action": "auth", "key": "{KEY_ID}", "secret": "{SECRET}"}
```

Keep in mind though, that you only have 10 seconds to do so after connecting.

If you provided correct credentials you will receive another success message:

JSON
```
[{"T":"success","msg":"authenticated"}]
```

Subscription

Congratulations, you are ready to receive real-time market data!

You can send one or more subscription messages. The general format of the subscribe message is this:

JSON
```
{
  "action": "subscribe",
  "channel1": ["SYMBOL1"],
  "channel2": ["SYMBOL2","SYMBOL3"],
  "channel3": ["*"]
}
```

You can subscribe to a particular symbol or to every symbol using the * wildcard. A subscribe message should contain what subscription you want to add to your current subscriptions in your session so you don't have to send what you're already subscribed to.
For example in the test stream, you can send this message:

JSON
```
{"action":"subscribe","trades":["FAKEPACA"]}
```

The available channels are described for each streaming endpoints separately.

Much like subscribe you can also send an unsubscribe message that subtracts the list of subscriptions specified from your current set of subscriptions.

JSON
```
{"action":"unsubscribe","quotes":["FAKEPACA"]}
```

After subscribing or unsubscribing you will receive a message that describes your current list of subscriptions.

JSON
```
[{"T":"subscription","trades":["AAPL"],"quotes":["AMD","CLDR"],"bars":["*"],"updat
edBars":[],"dailyBars":["VOO"],"statuses":["*"],"lulds":[],"corrections":["AAPL"],
"cancelErrors":["AAPL"]}]
```

You will always receive your entire list of subscriptions, as illustrated by the sample communication excerpt below:

JSON
```
> {"action": "subscribe", "trades": ["AAPL"], "quotes": ["AMD", "CLDR"], "bars":
["*"]}
<
[{"T":"subscription","trades":["AAPL"],"quotes":["AMD","CLDR"],"bars":["*"],"updat
edBars":[],"dailyBars":[],"statuses":[],"lulds":[],"corrections":["AAPL"],"cancelE
rrors":["AAPL"]}]
...
> {"action": "unsubscribe", "bars": ["*"]}
<
[{"T":"subscription","trades":["AAPL"],"quotes":["AMD","CLDR"],"bars":[],"updatedB
ars":[],"dailyBars":[],"statuses":[],"lulds":[],"corrections":["AAPL"],"cancelErro
rs":["AAPL"]}]
```

Messages

Format

Every message you receive from the server will be in the format:

json
```
[{"T": "{message_type}", {contents}},...]
```

Control messages (i.e. where `T` is `error`, `success` or `subscription`) always arrive in arrays of size one to make their processing easier.
Data points however may arrive in arrays that have a length that is greater than one. This is to facilitate clients whose connection is not fast enough to handle data points sent one by one. Our server buffers the outgoing messages but slow clients may get disconnected if their buffer becomes full.

Content type

You can use the `Content-Type` header to switch between text and binary message data frame:

- `Content-Type: application/json`
- `Content-Type: application/msgpack`

Encoding and Compression

Messages over the websocket are in encoded as clear text.

To reduce bandwidth requirements we have implemented compression as per RFC-7692. Our SDKs handle this for you so in most cases you won't have to implement anything yourself.

Errors

You may receive an error during your session. Below are all the errors you may run into.

Error Message	Description
`[{"T":"error","code":400,"msg":"invalid syntax"}]`	The message you sent to the server did not follow the specification. ⚠️ This can also be sent if the symbol in your

Error Message	Description
	subscription message is in invalid format.
`[{"T":"error","code":401,"msg":"not authenticated" }]`	You have attempted to subscribe or unsubscribe before authentication.
`[{"T":"error","code":402,"msg":"auth failed"}]`	You have provided invalid authentication credentials.
`[{"T":"error","code":403,"msg":"already authenticated"}]`	You have already successfully authenticated during your current session.
`[{"T":"error","code":404,"msg":"auth timeout"}]`	You failed to successfully authenticate after connecting. You have a few seconds to authenticate after connecting.
`[{"T":"error","code":405,"msg":"symbol limit exceeded"}]`	The symbol subscription request you sent would put you over the limit set by your subscription package. If this happens your symbol subscriptions are the same as they were before you sent the request that failed.
`[{"T":"error","code":406,"msg":"connection limit exceeded"}]`	You already have an ongoing authenticated session.
`[{"T":"error","code":407,"msg":"slow client"}]`	You may receive this if you are too slow to process the messages sent by the server. Please note that this is not guaranteed to arrive before you are disconnected to avoid keeping slow connections active forever.
`[{"T":"error","code":409,"msg":"insufficient subscription"}]`	You have attempted to access a data source not available in your subscription package.

Error Message	Description
`[{"T":"error","code":410,"msg":"invalid subscribe action for this feed"}]`	You tried to subscribe to channels not available in the stream, for example to `bars` in the option stream or to `trades` in the news stream.
`[{"T":"error","code":500,"msg":"internal error"}]`	An unexpected error occurred on our end. Please let us know if this happens.

Real-time Stock Data

Suggest Edits

This API provides stock market data on a websocket stream. This helps receive the most up to date market information that could help your trading strategy to act upon certain market movements. If you wish to access the latest pricing data, using the stream provides much better accuracy and performance than polling the latest historical endpoints.

You can find the general description of the real-time WebSocket Stream here. This page focuses on the stock stream.

URL

The URL for the stock stream is

`wss://stream.data.alpaca.markets/v2/{feed}`

Sandbox URL:

`wss://stream.data.sandbox.alpaca.markets/v2/{feed}`

Substitute `iex` or `sip` to `{feed}` depending on your subscription. The difference between SIP and IEX is described here.

Any attempt to access a data feed not available for your subscription will result in an error during authentication.

Channels

You can subscribe to the channels described in this section. For example

JSON

```
{"action":"subscribe","trades":["AAPL"],"quotes":["AMD","CLDR"],"bars":["*"]}
```

Trades

Schema

Attribute	Type	Notes
`T`	string	message type, always “t”
`S`	string	symbol
`i`	int	trade ID
`x`	string	exchange code where the trade occurred
`p`	number	trade price
`s`	int	trade size
`c`	array	trade condition
`t`	string	RFC-3339 formatted timestamp with nanosecond precision
`z`	string	tape

Example

JSON

```
{
  "T": "t",
  "i": 96921,
  "S": "AAPL",
  "x": "D",
  "p": 126.55,
  "s": 1,
  "t": "2021-02-22T15:51:44.208Z",
  "c": ["@", "I"],
  "z": "C"
}
```

Quotes

Schema

Attribute	Type	Notes
`T`	string	message type, always “q”
`S`	string	symbol
`ax`	string	ask exchange code

Attribute	Type	Notes
`ap`	number	ask price
`as`	int	ask size
`bx`	string	bid exchange code
`bp`	number	bid price
`bs`	int	bid size
`c`	array	quote condition
`t`	string	RFC-3339 formatted timestamp with nanosecond precision
`z`	string	tape

Example

JSON

```
{
  "T": "q",
  "S": "AMD",
  "bx": "U",
  "bp": 87.66,
  "bs": 1,
  "ax": "Q",
  "ap": 87.68,
  "as": 4,
  "t": "2021-02-22T15:51:45.335689322Z",
  "c": ["R"],
  "z": "C"
}
```

Bars

There are three separate channels where you can stream trade aggregates (bars).

Minute Bars (`bars`)

Minute bars are emitted right after each minute mark. They contain the trades from the previous minute. Trades from pre-market and aftermarket are also aggregated and sent out on the bars channel.

Daily Bars (`dailyBars`)

Daily bars are emitted right after each minute mark after the market opens. The daily bars contain all trades until the time they were emitted.

Updated Bars (`updatedBars`)

Updated bars are emitted after each half-minute mark if a "late" trade arrived after the previous minute mark. For example if a trade with a timestamp of `16:49:59.998` arrived right after `16:50:00`, just after `16:50:30` an updated bar with `t` set to `16:49:00` will be sent containing that trade, possibly updating the previous bar's closing price and volume.

Schema

Attribute	Type	Description
`T`	string	message type: "b", "d" or "u"
`S`	string	symbol
`o`	number	open price
`h`	number	high price
`l`	number	low price
`c`	number	close price
`v`	int	volume
`t`	string	RFC-3339 formatted timestamp

Example

JSON

```
{
  "T": "b",
  "S": "SPY",
  "o": 388.985,
  "h": 389.13,
  "l": 388.975,
  "c": 389.12,
  "v": 49378,
  "t": "2021-02-22T19:15:00Z"
}
```

Trade Corrections

These messages indicate that a previously sent trade was incorrect and they contain the corrected trade.

Subscription to trade corrections and cancel/errors is automatic when you subscribe to the trade channel.

{"action":"subscribe","trades":["AAPL"]}
[{"T":"subscription","trades":["AAPL"],"quotes":[],"bars":[],"updatedBars":[],"dai
lyBars":[],"statuses":[],"lulds":[],
"corrections":["AAPL"],"cancelErrors":["AAPL"]}]

Schema

Attribute	Type	Description
`T`	string	message type, always “c”
`S`	string	symbol
`x`	string	exchange code
`oi`	int	original trade id
`op`	number	original trade price
`os`	string	original trade size
`oc`	array	original trade conditions
`ci`	int	corrected trade id
`cp`	number	corrected trade price
`cs`	int	corrected trade size
`cc`	array	corrected trade conditions
`t`	string	RFC-3339 formatted timestamp
`z`	string	tape

Example

JSON


```
{
  "T": "c",
  "S": "EEM",
  "x": "M",
  "oi": 52983525033527,
  "op": 39.1582,
  "os": 440000,
  "oc": [
    " ",
    "7"
  ],
  "ci": 52983525034326,
  "cp": 39.1809,
  "cs": 440000,
  "cc": [
    " ",
```

```
    "7"
  ],
  "z": "B",
  "t": "2023-04-06T14:25:06.542305024Z"
}
```


Trade Cancels/Errors

These messages indicate that a previously sent trade was canceled.

Subscription to trade corrections and cancel/errors is automatic when you subscribe to the trade channel.


```
{"action":"subscribe","trades":["AAPL"]}
[{"T":"subscription","trades":["AAPL"],"quotes":[],"bars":[],"updatedBars":[],"dai
lyBars":[],"statuses":[],"lulds":[],
"corrections":["AAPL"],"cancelErrors":["AAPL"]}]
```


Schema

Attribute	Type	Description
`T`	string	message type, always “x”
`S`	string	symbol
`i`	int	trade id
`x`	string	trade exchange
`p`	number	trade price
`s`	int	trade size
`a`	string	action (“C” for cancel, “E” for error)
`t`	string	RFC-3339 formatted timestamp
`z`	string	tape

Example

JSON


```
{
  "T": "x",
  "S": "GOOGL",
  "i": 465,
  "x": "D",
  "p": 105.31,
  "s": 300,
  "a": "C",
  "z": "C",
```

```
  "t": "2023-04-06T13:15:42.83540958Z"
}
```


LULDs

Limit Up - Limit Down messages provide upper and lower limit price bands to securities.

Schema

Attribute	Type	Description
`T`	string	message type, always “l”
`S`	string	symbol
`u`	number	limit up price
`d`	number	limit down price
`i`	string	indicator
`t`	string	RFC-3339 formatted timestamp
`z`	string	tape

Example

JSON


```
{
  "T": "l",
  "S": "IONM",
  "u": 3.24,
  "d": 2.65,
  "i": "B",
  "t": "2023-04-06T13:34:45.565004401Z",
  "z": "C"
}
```


Trading Status

Identifies the trading status applicable to the security and reason for the trading halt if any. The status messages can be accessed from any {source} depending on your subscription.

To enable market data on a production environment please reach out to our sales team.

Schema

Attribute	Type	Description
`T`	string	message type, always “s”
`S`	string	symbol
`sc`	string	status code
`sm`	string	status message
`rc`	string	reason code
`rm`	string	reason message
`t`	string	RFC-3339 formatted timestamp
`z`	string	tape

Example

JSON

```
{
  "T": "s",
  "S": "AAPL",
  "sc": "H",
  "sm": "Trading Halt",
  "rc": "T12",
  "rm": "Trading Halted; For information requested by NASDAQ",
  "t": "2021-02-22T19:15:00Z",
  "z": "C"
}
```

Status Codes

Tape A & B (CTA)

Code	Value
2	Trading Halt
3	Resume
5	Price Indication
6	Trading Range Indication
7	Market Imbalance Buy
8	Market Imbalance Sell

Code	Value
9	Market On Close Imbalance Buy
A	Market On Close Imbalance Sell
C	No Market Imbalance
D	No Market On Close Imbalance
E	Short Sale Restriction
F	Limit Up-Limit Down

Tape C & O (UTP)

Codes	Resume
H	Trading Halt
Q	Quotation Resumption
T	Trading Resumption
P	Volatility Trading Pause

Readon Codes

Tape A & B (CTA)

Code	Value
D	News Released (formerly News Dissemination)
I	Order Imbalance
M	Limit Up-Limit Down (LULD) Trading Pause
P	News Pending
X	Operational
Y	Sub-Penny Trading
1	Market-Wide Circuit Breaker Level 1 – Breached
2	Market-Wide Circuit Breaker Level 2 – Breached
3	Market-Wide Circuit Breaker Level 3 – Breached

Tape C & O (UTP)

Code	Value
T1	Halt News Pending
T2	Halt News Dissemination
T5	Single Stock Trading Pause In Affect
T6	Regulatory Halt Extraordinary Market Activity
T8	Halt ETF
T12	Trading Halted; For information requested by NASDAQ
H4	Halt Non Compliance
H9	Halt Filings Not Current
H10	Halt SEC Trading Suspension
H11	Halt Regulatory Concern
01	Operations Halt, Contact Market Operations
IPO1	IPO Issue not yet Trading
M1	Corporate Action
M2	Quotation Not Available
LUDP	Volatility Trading Pause
LUDS	Volatility Trading Pause – Straddle Condition
MWC1	Market Wide Circuit Breaker Halt – Level 1
MWC2	Market Wide Circuit Breaker Halt – Level 2
MWC3	Market Wide Circuit Breaker Halt – Level 3
MWC0	Market Wide Circuit Breaker Halt – Carry over from previous day
T3	News and Resumption Times
T7	Single Stock Trading Pause/Quotation-Only Period
R4	Qualifications Issues Reviewed/Resolved; Quotations/Trading to Resume
R9	Filing Requirements Satisfied/Resolved; Quotations/Trading To Resume
C3	Issuer News Not Forthcoming; Quotations/Trading To Resume
C4	Qualifications Halt ended; maint. Req. met; Resume
C9	Qualifications Halt Concluded; Filings Met; Quotes/Trades To Resume

Code	Value
C11	Trade Halt Concluded By Other Regulatory Auth,; Quotes/Trades Resume
R1	New Issue Available
R	Issue Available
IPOQ	IPO security released for quotation
IPOE	IPO security – positioning window extension
MWCQ	Market Wide Circuit Breaker Resumption

Example

```
$ wscat -c wss://stream.data.alpaca.markets/v2/sip
connected (press CTRL+C to quit)
< [{"T":"success","msg":"connected"}]
> {"action": "auth", "key": "******", "secret": "******"}
< [{"T":"success","msg":"authenticated"}]
> {"action": "subscribe", "trades": ["AAPL"], "quotes": ["AMD", "CLDR"], "bars":
["*"],"dailyBars":["VOO"],"statuses":["*"]}
<
[{"T":"subscription","trades":["AAPL"],"quotes":["AMD","CLDR"],"bars":["*"],"updat
edBars":[],"dailyBars":["VOO"],"statuses":["*"],"lulds":[],"corrections":["AAPL"],
"cancelErrors":["AAPL"]}]
<
[{"T":"q","S":"AMD","bx":"K","bp":91.95,"bs":2,"ax":"Q","ap":91.98,"as":1,"c":["R"
],"z":"C","t":"2023-04-06T11:54:21.670905508Z"}]
<
[{"T":"t","S":"AAPL","i":628,"x":"K","p":162.92,"s":3,"c":["@","F","T","I"],"z":"C
","t":"2023-04-
06T11:54:26.838232225Z"},{"T":"t","S":"AAPL","i":75,"x":"Z","p":162.92,"s":3,"c":[
"@","F","T","I"],"z":"C","t":"2023-04-
06T11:54:26.838562809Z"},{"T":"t","S":"AAPL","i":1465,"x":"P","p":162.91,"s":71,"c
":["@","F","T","I"],"z":"C","t":"2023-04-06T11:54:26.83915973Z"}]
<
[{"T":"q","S":"AMD","bx":"P","bp":91.9,"bs":1,"ax":"Q","ap":91.98,"as":1,"c":["R"]
,"z":"C","t":"2023-04-06T11:54:27.924933876Z"}]
```

Real-time Crypto Data

Suggest Edits

Crypto Data API provides websocket streaming for trades, quotes, orderbooks, minute bars and daily bars. This helps receive the most up to date market information that could help your trading strategy to act upon certain market movements.

Since Alpaca now executes your crypto orders in its own exchange, the `v1beta3` crypto market data endpoints no longer distribute data from other providers, but from Alpaca itself.

You can find the general description of the real-time WebSocket Stream here. This page focuses on the crypto stream.

Advanced Websockets Tutorial

Check out our tutorial Advanced Live Websocket Crypto Data Streams in Python for some tips on handling live crypto data stream in Python.

URL

The URL for the crypto stream is

`wss://stream.data.alpaca.markets/v1beta3/crypto/us`

Sandbox URL:

`wss://stream.data.sandbox.alpaca.markets/v1beta3/crypto/us`

Multiple data points may arrive in each message received from the server. These data points have the following formats, depending on their type.

Channels

Trades

Schema

Attribute	Type	Notes
`T`	string	message type, always “t”
`S`	string	symbol
`p`	number	trade price
`s`	number	trade size
`t`	string	RFC-3339 formatted timestamp with nanosecond precision
`i`	int	trade ID
`tks`	string	taker side: B for buyer, S for seller

Example

JSON


```
{
  "T": "t",
  "S": "AVAX/USD",
  "p": 47.299,
  "s": 29.205707815,
  "t": "2024-03-12T10:27:48.858228144Z",
  "i": 3447222699101865076,
  "tks": "S"
}
```


Quotes

Schema

Attribute	Type	Notes
`T`	string	message type, always “q”
`S`	string	symbol
`bp`	number	bid price
`bs`	number	bid size
`ap`	number	ask price
`as`	number	ask size
`t`	string	RFC-3339 formatted timestamp with nanosecond precision

Example

JSON


```
{
  "T": "q",
  "S": "BAT/USD",
  "bp": 0.35718,
  "bs": 13445.46,
  "ap": 0.3581,
  "as": 13561.902,
  "t": "2024-03-12T10:29:43.111588173Z"
}
```


Bars

Crypto bars contain quote mid-prices

Due to the volatility of some currencies, including lack of trade volume at any given time, we include the quote midpoint prices in the bars to offer a better data experience. If in a bar no trade happens, the volume will be 0, but the prices will be determined by the quote prices.

There are three separate channels where you can stream trade aggregates (bars).

Minute Bars (`bars`)

Minute bars are emitted right after each minute mark. They contain the trades and quote midpoints from the previous minute.

Daily Bars (`dailyBars`)

Daily bars are emitted right after each minute mark after the market opens. The daily bars contain all trades and quote midprices until the time they were emitted.

Updated Bars (`updatedBars`)

Updated bars are emitted after each half-minute mark if a "late" trade arrived after the previous minute mark. For example if a trade with a timestamp of `16:49:59.998` arrived right after `16:50:00`, just after `16:50:30` an updated bar with `t` set to `16:49:00` will be sent containing that trade, possibly updating the previous bar's closing price and volume.

Schema

Attribute	Type	Description
`T`	string	message type: "b", "d" or "u"
`S`	string	symbol
`o`	number	open price
`h`	number	high price
`l`	number	low price
`c`	number	close price
`v`	int	volume
`t`	string	RFC-3339 formatted timestamp

Example

JSON


```
{
  "T": "b",
  "S": "BTC/USD",
  "o": 71856.1435,
  "h": 71856.1435,
  "l": 71856.1435,
  "c": 71856.1435,
  "v": 0,
  "t": "2024-03-12T10:37:00Z",
  "n": 0,
  "vw": 0
}
```


Orderbooks

Schema

Attribute	Type	Notes
`T`	string	message type, always “o”
`S`	string	symbol
`t`	string	RFC-3339 formatted timestamp with nanosecond precision
`b`	array	bids: array of `p` (price) and `s` pairs. If `s` is zero, it means that that bid entry was remov the orderbook. Otherwise it was added or updated.
`a`	array	asks: array of `p` (price) and `s` pairs. If `s` is zero, it means that that ask entry was remo the orderbook. Otherwise it was added or updated.
`r`	boolean	reset: if true, the orderbook message contains the whole server side orderbook. This to the client that they should reset their orderbook. Typically sent as the first messag subscription.

Example

Initial full orderbook

JSON


```
{
  "T": "o",
  "S": "BTC/USD",
  "t": "2024-03-12T10:38:50.79613221Z",
  "b": [
    {
      "p": 71859.53,
      "s": 0.27994
```

```
    },
    {
      "p": 71849.4,
      "s": 0.553986
    },
    {
      "p": 71820.469,
      "s": 0.83495
    },
   ...
  ],
  "a": [
    {
      "p": 71939.7,
      "s": 0.83953
    },
    {
      "p": 71940.4,
      "s": 0.28025
    },
    {
      "p": 71950.715,
      "s": 0.555928
    },
    ...
  ],
  "r": true
}
```


`r` is true, meaning that this message contains the whole BTC/USD orderbook. It's truncated here for readability, the actual book has a lot more bids & asks.

Update message

json


```
{
  "T": "o",
  "S": "MKR/USD",
  "t": "2024-03-12T10:39:39.445492807Z",
  "b": [],
  "a": [
    {
      "p": 2614.587,
      "s": 12.5308
    }
  ]
}
```


This means that the ask price level 2614.587 was changed to 12.5308. If there were previously no 2614.587 ask entry in the orderbook, then it should be added, if there were, its size should be updated.

Remove message

JSON

```
{
  "T": "o",
  "S": "CRV/USD",
  "t": "2024-03-12T10:39:40.501160019Z",
  "b": [
    {
      "p": 0.7904,
      "s": 0
    }
  ],
  "a": []
}
```


This means that the 0.7904 bid price level should be removed from the orderbook.

Example

```
$ wscat -c wss://stream.data.alpaca.markets/v1beta3/crypto/us
connected (press CTRL+C to quit)
< [{"T":"success","msg":"connected"}]
> {"action": "auth", "key": "**\***", "secret": "**\***"}
< [{"T":"success","msg":"authenticated"}]
> {"action": "subscribe", "bars": ["BTC/USD"]}
<
[{"T":"subscription","trades":[],"quotes":[],"orderbooks":[],"bars":["BTC/USD"],"u
pdatedBars":[],"dailyBars":[]}]
<
[{"T":"b","S":"BTC/USD","o":26675.04,"h":26695.36,"l":26668.79,"c":26688.7,"v":3.2
27759152,"t":"2023-03-17T12:28:00Z","n":93,"vw":26679.5912436798}]
<
[{"T":"b","S":"BTC/USD","o":26687.9,"h":26692.91,"l":26628.55,"c":26651.39,"v":11.
568622108,"t":"2023-03-17T12
```

Real-time News

Suggest Edits

This API provides stock market news on a websocket stream. You can find the general description of the real-time WebSocket Stream here. This page focuses on the news stream.

URL

The URL for the news stream is

```
wss://stream.data.alpaca.markets/v1beta1/news
```

Sandbox URL:

```
wss://stream.data.sandbox.alpaca.markets/v1beta1/news
```

Channels

News

Schema

Attribute	Type	Notes
T	string	Type of message (“n” for news)
id	int	News article ID
headline	string	Headline or title of the article
summary	string	Summary text for article (may be first sentence of content)
author	string	Original author of news article
created_at	string	Date article was created in RFC-3339 format
updated_at	string	Date article was updated in RFC-3339 format
content	string	Content of news article (might contain HTML)
url	string	URL of article (if applicable)
symbols	array	List of related or mentioned symbols
source	string	Source where the news originated from (e.g. Benzinga)

Example

JSON


```
{
    "T": "n",
    "id": 24918784,
    "headline": "Corsair Reports Purchase Of Majority Ownership In iDisplay, No
Terms Disclosed",
    "summary": "Corsair Gaming, Inc. (NASDAQ:CRSR) (“Corsair”), a leading global
provider and innovator of high-performance gear for gamers and content creators,
today announced that it acquired a 51% stake in iDisplay",
    "author": "Benzinga Newsdesk",
    "created_at": "2022-01-05T22:00:37Z",
    "updated_at": "2022-01-05T22:00:38Z",
    "url": "https://www.benzinga.com/m-a/22/01/24918784/corsair-reports-purchase-
of-majority-ownership-in-idisplay-no-terms-disclosed",
    "content": "\u003cp\u003eCorsair Gaming, Inc. (NASDAQ:\u003ca class=\"ticker\"
href=\"https://www.benzinga.com/stock/CRSR#NASDAQ\"\u003eCRSR\u003c/a\u003e)
(\u0026ldquo;Corsair\u0026rdquo;), a leading global ...",
    "symbols": ["CRSR"],
    "source": "benzinga"
}
```


Real-time Option Data

Suggest Edits

This API provides option market data on a websocket stream. This helps receive the most up to date market information that could help your trading strategy to act upon certain market movements. If you wish to access the latest pricing data, using the stream provides much better accuracy and performance than polling the latest historical endpoints.

You can find the general description of the real-time WebSocket Stream here. This page focuses on the option stream.

URL

The URL for the option stream is

`wss://stream.data.alpaca.markets/v1beta1/{feed}`
Sandbox URL:

`wss://stream.data.sandbox.alpaca.markets/v1beta1/{feed}`
Substitute `indicative` or `opra` to `{feed}` depending on your subscription. The difference between the two feeds is described here.
Any attempt to access a data feed not available for your subscription will result in an error during authentication.

Message format

Msgpack

Unlike the stock and crypto stream, the option stream is only available in msgpack format. The SDKs are using this format automatically. For readability, the examples in the rest of this documentation will still be in json format (because msgpack is binary encoded).

Channels

Trades

Schema

Attribute	Type	Notes
`T`	string	message type, always “t”
`S`	string	symbol
`t`	string	RFC-3339 formatted timestamp with nanosecond precision
`p`	number	trade price
`s`	int	trade size
`x`	string	exchange code where the trade occurred
`c`	string	trade condition

Example

JSON

```
{
  "T": "t",
  "S": "AAPL240315C00172500",
  "t": "2024-03-11T13:35:35.13312256Z",
  "p": 2.84,
  "s": 1,
  "x": "N",
  "c": "S"
}
```

Quotes

Schema

Attribute	Type	Notes
`T`	string	message type, always “q”
`S`	string	symbol
`t`	string	RFC-3339 formatted timestamp with nanosecond precision
`bx`	string	bid exchange code
`bp`	number	bid price
`bs`	int	bid size
`ax`	string	ask exchange code
`ap`	number	ask price
`as`	int	ask size

Attribute	Type	Notes
`c`	string	quote condition

Example

JSON

```
{
  "T": "q",
  "S": "SPXW240327P04925000",
  "t": "2024-03-12T11:59:38.897261568Z",
  "bx": "C",
  "bp": 9.46,
  "bs": 53,
  "ax": "C",
  "ap": 9.66,
  "as": 38,
  "c": "A"
}
```

Errors

Other than the general stream errors, you may receive these option-specific errors during your session:

Error Message	Description
`[{"T":"error","code":412,"msg":"option messages are only available in MsgPack format"}]`	Use the `Content-Type: application/msgpack` header.
`[{"T":"error","code":413,"msg":"star subscription is not allowed for option quotes"}]`	You cannot subscribe to `*` for option qu (there are simply too many of them).

Updated 15 days ago

Market Data FAQ

Frequently Asked Questions

Suggest Edits

Stocks

What's the difference between IEX and SIP data?

SIP is short for Securities Information Processor. All US exchanges are mandated by the regulators to report their activities (trades and quotes) to the consolidated tape. This is what we call SIP data.

IEX (Investors Exchange) is a single stock exchange.

Websocket stream

Our free market data offering includes live data only from the IEX exchange:

`wss://stream.data.alpaca.markets/v2/iex`

The Algo Trader Plus subscription on the other hand offers SIP data:

`wss://stream.data.alpaca.markets/v2/sip`

Historical data

On the historical endpoints you can use the `feed` parameter to switch between the two data feeds:

JSON

```
$ curl -s -H "APCA-API-KEY-ID: ${APCA_API_KEY_ID}" -H "APCA-API-SECRET-KEY:
${APCA_API_SECRET_KEY}" \

"https://data.alpaca.markets/v2/stocks/AAPL/bars?feed=sip&timeframe=1Day&start=202
3-09-29&limit=1" | jq .
{
  "bars": [
    {
      "t": "2023-09-29T04:00:00Z",
      "o": 172.02,
      "h": 173.07,
      "l": 170.341,
      "c": 171.21,
      "v": 51861083,
      "n": 535134,
      "vw": 171.599691
    }
  ],
  "symbol": "AAPL",
  "next_page_token": "QUFQTHxEfDIwMjMtMDktMjlUMDQ6MDA6MDAuMDAwMDAwMDAwWg=="
}
$ curl -s -H "APCA-API-KEY-ID: ${APCA_API_KEY_ID}" -H "APCA-API-SECRET-KEY:
${APCA_API_SECRET_KEY}" \

"https://data.alpaca.markets/v2/stocks/AAPL/bars?feed=iex&timeframe=1Day&start=202
3-09-29&limit=1" | jq .
{
  "bars": [
    {
      "t": "2023-09-29T04:00:00Z",
      "o": 172.015,
      "h": 173.06,
      "l": 170.36,
      "c": 171.29,
      "v": 923134,
      "n": 12630,
      "vw": 171.716432
    }
  ],
```

```
  "symbol": "AAPL",
  "next_page_token": null
}
```

In this example (2023-09-29 Apple daily bar) you can clearly see the difference between the two feeds: there were **12 630** eligible trades on the IEX exchange that day and more than **535 136** trade in total across all exchanges (naturally including IEX). Similar difference can be seen between the volumes.

All the latest endpoints (including the snapshot endpoint) require a subscription to be used with the SIP feed. And for historical queries, the `end` parameter must be at least 15 minutes old to query SIP data without a subscription. The default value for `feed` is always the "best" available feed based on the user's subscription.
JSON
```
$  curl -s -H "APCA-API-KEY-ID: ${APCA_API_KEY_ID}" -H "APCA-API-SECRET-KEY:
${APCA_API_SECRET_KEY}" \
  "https://data.alpaca.markets/v2/stocks/AAPL/trades/latest" | jq .
{
  "symbol": "AAPL",
  "trade": {
    "t": "2023-09-29T19:59:59.246196362Z",
    "x": "V",  // << IEX exchange code
    "p": 171.29,
    "s": 172,
    "c": [
      "@"
    ],
    "i": 12727,
    "z": "C"
  }
}
$ curl -H "APCA-API-KEY-ID: ${APCA_API_KEY_ID}" -H "APCA-API-SECRET-KEY:
${APCA_API_SECRET_KEY}" \
  "https://data.alpaca.markets/v2/stocks/AAPL/trades/latest?feed=sip"
{"code":42210000,"message":"subscription does not permit querying recent SIP
data"}
```

In this example we're querying the latest AAPL trade without a subscription. We can see that the default `feed` in this case is `iex`. If we try to query the SIP feed, we get an error. To fix that error, we need to subscribe to Algo Trader Plus.

Why can't I find market data for a particular symbol (e.g. CGRNQ)?

OTC: Make sure the symbol is not traded in OTC using the assets endpoint `https://api.alpaca.markets/v2/assets/CGRNQ` returns
JSON
```
{
  "id": "dc2d8be9-33b5-4a32-8f57-5b7d209d2c82",
  "class": "us_equity",
  "exchange": "OTC", // << This symbol is traded in OTC
  "symbol": "CGRNQ",
  "name": "CAPSTONE GREEN ENERGY CORP COM PAR $.001",
```

```
  "status": "active",
  "tradable": false,
  "marginable": false,
  "maintenance_margin_requirement": 100,
  "shortable": true,
  "easy_to_borrow": true,
  "fractionable": true,
  "attributes": []
}
```


Market data for OTC symbols can only be queried with a special subscription currently only available for broker partners. If you do have the subscription, you can use `feed=otc` to query the data.
Halt: Make sure the symbol isn't halted, or wasn't halted at the time you're querying. You can check the current halts or the historical halts on the Nasdaq website. For example the symbol SVA has been halted since 2019-02-22.

Crypto

Why are there crypto bars with 0 volume / trade count?

Our crypto market data reflects trades and quotes from our own Alpaca exchange. Due to the volatility of some currencies, including lack of trade volume at any given time, we include the quote midpoint prices to offer a better data experience. If in a bar no trade happens, the volume will be 0, but the prices will be determined by the quote prices.

About Connect API

Develop applications on Alpaca's platform using OAuth2. Let 4M+ users with an Alpaca brokerage account connect to your app.

Suggest Edits

Develop applications on Alpaca's platform using OAuth2. Alpaca's OAuth allows you to seamlessly integrate financial markets into your application and expand your audience to the over 100K brokerage accounts on Alpaca's platform.

Read Register Your App to learn how you can register your app. In addition, you can visit OAuth Integration Guide to learn more about using OAuth to connect your applications with Alpaca.

Broker Partners

Broker partners are able to create their own OAuth service. Allow your end users to use OAuth apps like TradingView through your Broker API application. Learn more about OAuth with Broker API in the Broker API reference

Terms of Access and Use

- You must read the terms and register in order to connect and use Alpaca's APIs
- All API clients must authenticate with OAuth 2.0
- You may not imply that the app was developed by Alpaca.
- If you are building a commercial application that makes money (including ads, in-app purchases, etc), you must disclose it in the registration form and receive written approval.
- To allow live trading for other users, the app needs to be approved by Alpaca. Please contact partnership@alpaca.markets.
- Live trading is allowed for the app developer user without approval.

!

This is not an offer, solicitation of an offer, or advice to open a brokerage account.

Disclosure can be found here

FAQs

Q: What can an OAuth app do?

A: OAuth allows you to manage your end-user's Alpaca brokerage account on their behalf. This means you can create many types of financial services including automated investing, portfolio analytics and much more.

Q: Should I use OAuth or Broker API?

A: OAuth allows you to expand your audience to users with Alpaca brokerage accounts. On the otherhand, Broker API allows you to build an application fully within your environment. Users sign up for a brokerage account under your application. If you want to create your own brokerage, automated investment app, or any app where you want to own your users, use the Broker API. If you want to build your trading service on Alpaca's platform, use OAuth.

Q: How secure is OAuth?

A: OAuth2 itself is very secure. However you must make sure to follow good practices in how you handle tokens. Make sure to never publicly expose your client secret and access tokens.

Q: How to get OAuth App live?

A: You will need to register your app in the OAuth apps section of the dashboard. Learn more about Register Your App.

Q: I'm developing an app/service targeting non-US users. Can we integrate with Alpaca's OAuth API?

A: Alpaca's platform supports brokerage accounts for international users. When you build an app on OAuth, all users on Alpaca's platform will be able to use your service, including international users.

Using OAuth2 and Trading API

Alpaca implements OAuth 2.0 to allow third party applications to access Alpaca Trading API on behalf of the end-users. This document describes how you can integrate with Alpaca through OAuth.

Suggest Edits

By default once you have a valid client_id and client_secret, any paper account and the live account associated with the OAuth Client will be available to connect to your app. We welcome developers to build applications and products that are powered by Alpaca while also protecting the privacy and security of our users. To build using Alpaca's APIs, please follow the guide below.

Getting the Access Token

At a high level the flow looks like this, we will go into detail about each step

1. User requests a connection between your application and Alpaca
2. User is redirected to Alpaca to login and authorize the application from inside the dashbaord
3. Alpaca grants an authorization token to your application thorugh user-agent
4. You application then makes an access token request
5. Alpaca returns an access token grant.

1. Request for Connection on Behalf of User

When redirecting a user to Alpaca to authorize access to your application, you'll need to construct the authorization URL with the correct parameters and scopes.

```
GET
https://app.alpaca.markets/oauth/authorize?response_type=code&client_id=YOUR_CLIEN
T_ID&redirect_uri=YOUR_REDIRECT_URL&state=SOMETHING_RANDOM&scope=account:write%20t
rading
```

Here's a list of parameters you should always specify:

Parameter	Required?	Description
`response_type`	Required	Must be `code` to request an authorization code.
`client_id`	Required	The `client_id` you were provided with when registering your app
`redirect_uri`	Required	The redirect URL where the user will be sent after authorization. It must ma the whitelisted redirect URIs for your application.
`state`	Optional	An unguessable random string, used to protect against request forgery attac
`scope`	Optional	A space-delimited list of scopes your application requests access to. Read-o endpoint access is assumed by default.

Allowed Scopes

Scope	Description
`account:write`	Write access for account configurations and watchlists.
`trading`	Place, cancel or modify orders.
`data`	Access to the Data API.

We will be adding more scopes soon.

2. Users Authorizing App to Access Alpaca Account

From the user side, they will see the following authorization screen

3. Alpaca Redirect Back to App

If the user approves access, Alpaca will redirect them back to your `redirect_uri` with a temporary `code` parameter. If you specified a state parameter in step 1, it will be returned as well. The parameter will always match the value specified in step 1. If the values don't match, the request should not be trusted.
Example

```
GET https://example.com/oauth/callback?code=67f74f5a-a2cc-4ebd-88b4-
22453fe07994&state=8e02c9c6a3484fadaaf841fb1df290e1
```

4. App Receives Authorization Code

You can then use this code to exchange for an access token.

5. App Exchanges Auth Code with Access Token

After you have received the temporary `code`, you can exchange it for an access token. This can be done by making a `POST` call to `https://api.alpaca.markets/oauth/token`

Parameters (All Required)

Parameter	Description
`grant_type`	Must be set to authorization_code for an access token request.
`code`	The authorization code received in step 4
`client_id`	The Client ID you received when you registered the application.
`client_secret`	The Client Secret you received when you registered the application.
`redirect_uri`	The redirect URI you used for the authorization code request.

Note

This request should take place behind-the-scenes from your backend server and shouldn't be visible to the end users for security purposes.

The content type must be application/x-www-form-urlencoded as defined in RFC.

Example request:

cURL

```
curl -X POST https://api.alpaca.markets/oauth/token \
  -d 'grant_type=authorization_code&code=67f74f5a-a2cc-4ebd-88b4-
22453fe07994&client_id=fc9c55efa3924f369d6c1148e668bbe8&client_secret=5b8027074d8a
b434882c0806833e76508861c366&redirect_uri=https://example.com/oauth/callback'
```

After a successful request, a valid access token will be returned in the response:

JSON

```
{
    "access_token": "79500537-5796-4230-9661-7f7108877c60",
    "token_type": "bearer",
    "scope": "account:write trading"
}
```

API Calls

Once you have integrated and have a valid access token you can start make calls to Alpaca Trading API v2 on behalf of the end-user.

Example Requests

A

cURL
```
curl https://api.alpaca.markets/v2/account /
  -H 'Authorization: Bearer 79500537-5796-4230-9661-7f7108877c60'
```

cURL
```
curl https://paper-api.alpaca.markets/v2/orders /
  -H 'Authorization: Bearer 79500537-5796-4230-9661-7f7108877c60'
```

The OAuth token can also be used for the trade update websockets stream.

```
{
  "action": "authenticate",
  "data": {
    "oauth_token": "79500537-5796-4230-9661-7f7108877c60"
  }
}
```

Getting Started

Suggest Edits

About

Alpaca-py provides an interface for interacting with the API products Alpaca offers. These API products are provided as various REST, WebSocket and SSE endpoints that allow you to do everything from streaming market data to creating your own trading apps.

Python SDK Docs

Usage

Alpaca's APIs allow you to do everything from building algorithmic trading strategies to building a full brokerage experience for your own end users. Here are some things you can do with Alpaca-py.

Market Data API: Access live and historical market data for 5000+ stocks and 20+ crypto.

Trading API: Trade stock and crypto with lightning fast execution speeds.

Broker API & Connect: Build investment apps - from robo-advisors to brokerages.

Installation

Alpaca-py is supported on Python 3.7+. You can install Alpaca-py using pip. To learn more about version histories, visit the PyPi page.

To install Alpaca-py, run the following pip command in your terminal.

Shell
```
pip install alpaca-py
```

Errors

Try upgrading your pip before installing if you face errors.

Shell
```
pip install --upgrade pip
```

Poetry

If you're using poetry to manage dependencies in your project. You can add Alpaca-py
to your project by running

Shell
```
poetry add alpaca-py
```

What's New?

If you've used the previous python SDK alpaca-trade-api, there are a few key differences to be aware of.

Broker API

Alpaca-py lets you use Broker API to start building your investment apps! Learn more at the
:ref:broker page.

OOP Design

Alpaca-py uses a more OOP approach to submitting requests compared to the previous SDK. To submit a request, you will most likely need to create a request object containing the desired request data. Generally, there is a unique request model for each method.

Some examples of request models corresponding to methods:

- `GetOrdersRequest` for `TradingClient.get_orders()`
- `CryptoLatestOrderbookRequest` for `CryptoHistoricalDataClient.get_crypto_latest_orderbook()`

Request Models Usage Example

To get historical bar data for crypto, you will need to provide a CryptoBarsRequest object.

Python
```python
from alpaca.data.historical import CryptoHistoricalDataClient
from alpaca.data.requests import CryptoBarsRequest
from alpaca.data.timeframe import TimeFrame

# no keys required for crypto data
client = CryptoHistoricalDataClient()

request_params = CryptoBarsRequest(
                        symbol_or_symbols=["BTC/USD", "ETH/USD"],
                        timeframe=TimeFrame.Day,
                        start="2022-07-01"
                 )

bars = client.get_crypto_bars(request_params)
```

Data Validation

Alpaca-py uses pydantic to validate data models at run-time. This means if you are receiving request data via JSON from a client. You can handle parsing and validation through Alpaca's request models. All request models can be

instantiated
by passing in data in dictionary format.

Here is a rough example of what is possible.

Python
```
@app.route('/post_json', methods=['POST'])
def do_trade():
    # ...

    order_data_json = request.get_json()

    # validate data
    MarketOrderRequest(**order_data_json)

    # ...
```

Many Clients

Alpaca-py has a lot of client classes. There is a client for each API and even asset class specific clients (`StockHistoricalData`, `CryptoDataStream`).
This requires you to pick and choose clients based on your needs.

- Broker API:
  - `BrokerClient`
- Trading API
  - `TradingClient`
- Market Data API:
  - `StockHistoricalDataClient`
  - `CryptoHistoricalDataClient`
  - `CryptoDataStream`
  - `StockDataStream`

API Keys

Trading and Market Data API

In order to use Alpaca's services you'll need to sign up for an Alpaca account and retrieve your API keys.
Signing up is completely free and takes only a few minutes. Sandbox environments are available to test
out the API.

To use the sandbox environment, you will need to provide sandbox/paper keys.
API keys are passed into Alpaca-py through

either `TradingClient`, `StockHistoricalDataClient`, `CryptoHistoricalDataClient`, `StockDataStream`, or `CryptoDataStream`.

Broker API

To use the Broker API, you will need to sign up for a broker account and retrieve your Broker API keys. The API keys can be found on the dashboard once you've logged in. Alpaca also provides a sandbox environment to test out Broker API.

To use the sandbox mode, provide your sandbox keys. Once you have your keys, you can pass them into `BrokerClient` to get started.

About Market Data API

Gain seamless access to a wealth of data with Alpaca Market Data API, offering real-time and historical equities & crypto information.

Suggest Edits

Overview

The Market Data API v2 offers seamless access to market data through both HTTP and WebSocket protocols. With a focus on historical and real-time data, developers can efficiently integrate these APIs into their applications.

To simplify the integration process, we provide user-friendly SDKs in Python, Go, NodeJS, and C#. These SDKs offer comprehensive functionalities, making it easier for developers to work with the Market Data APIs & Web Sockets.

To start using the APIs, developers have two convenient options: they can either access it through the public workspace on Postman or directly from our GitHub repository.

By leveraging Alpaca Market Data API v2 and its associated SDKs, developers can seamlessly incorporate historical and real-time market data into their applications, enabling them to build powerful and data-driven financial products.

Subscription Plans

The Market Data API v2 offers Market Data access under three distinct plans: **Free**, **Algo Trader Plus**, and **Broker Professional**.

The Free plan serves as the default option for both Paper and Live trading accounts, ensuring all users can access essential data with zero cost.

In addition to the Free plan, we also provide two premium options: Algo Trader Plus and Broker Professional. These plans cater to the needs of traders and brokers who require advanced data features and increased data usage allowances.

Equities

	Free	**Algo Trader Plus**	**Broker Professional**
Pricing	Free	$99/month	$99/mo/device
Securities coverage	US Stocks & ETFs	US Stocks & ETFs	US Stocks & ETFs
Real-time market coverage	IEX	All US Stock Exchanges	All US Stock Exchanges
Websocket subscriptions	30 symbols	Unlimited	Unlimited
Historical data timeframe	Since 2016	Since 2016	Since 2016
Historical data limitation*	latest 15 minutes	no restriction	no restriction
Historical API calls	200/min	10,000/min	10,000/min

Our data sources are directly fed by the CTA (Consolidated Tape Association), which is administered by NYSE (New York Stock Exchange), and the UTP (Unlisted Trading Privileges) stream, which is administered by Nasdaq. The synergy of these two sources ensures comprehensive market coverage, encompassing 100% of market volume.

Options

	Free	Algo Trader Plus
Securities coverage	US Options Securities	US Options Securities
Real-time market coverage	Indicative Pricing Feed	OPRA Feed
Websocket subscriptions	200 quotes	1000 quotes
Historical data limitation*	latest 15 minutes	no restriction
Historical API calls	200/min	10,000/min

Our options data sources are directly fed by OPRA (Options Price Reporting Authority).

Free vs Algo Trader Plus*

1. Users on the Free Market Data plan have access to the IEX (Investors Exchange LLC) stream but with some limitations. Historical equities data is available without restrictions, however, it is not allowed to query the latest 15-minute data.
2. This also means that users on the Free Market Data plan may not be able to access the most recent SIP (Securities Information Processor) data or use certain latest endpoints that rely exclusively on SIP data.
3. Moreover, users on the Free Market Data plan can access Alpaca's Indicative Pricing Feed for free. The same limitation applies to historical options data as equities (IEX), users can only query more than 15-minute old data.

Broker Professional

We offer custom pricing and tailored solutions for Broker API partners seeking to leverage our comprehensive market data. Our goal is to meet the specific needs and requirements of our valued partners, ensuring they have access to the data and tools necessary to enhance their services and provide exceptional value to their customers.

For detailed information about our pricing options and the benefits of becoming a Broker API partner, kindly reach out to our sales team.

Exchanges

Alpaca supports a range of stock exchanges, each identified by a unique tape ID, which is returned in all market data requests. To help you easily map the tape code to the corresponding exchange, here is the list:

Exchange Code	Exchange Name
A	NYSE American (AMEX)
B	NASDAQ OMX BX
C	National Stock Exchange
D	FINRA ADF
E	Market Independent
H	MIAX
I	International Securities Exchange
J	Cboe EDGA
K	Cboe EDGX
L	Long Term Stock Exchange
M	Chicago Stock Exchange
N	New York Stock Exchange
P	NYSE Arca
Q	NASDAQ OMX
S	NASDAQ Small Cap
T	NASDAQ Int
U	Members Exchange
V	IEX
W	CBOE
X	NASDAQ OMX PSX
Y	Cboe BYX
Z	Cboe BZX

Conditions

Every feed or stock exchange utilizes its unique set of codes to identify trade and quote conditions. Consequently, the same condition may vary in code representation depending on the data's originator.

Trade conditions

CTS

Below is a table containing codes that indicate specific trade conditions applicable under the CTA (Consolidated Tape Association) Plan:

For further details on this topic, please refer to page 64 of the Consolidated Tape System (CTS) Specification.

Code	Value
`Space`	Regular Sale
B	Average Price Trade
C	Cash Trade (Same Day Clearing)
E	Automatic Execution
F	Inter-market Sweep Order
H	Price Variation Trade
I	Odd Lot Trade
K	Rule 127 (NYSE only) or Rule 155 (NYSE MKT only)
L	Sold Last (Late Reporting)
M	Market Center Official Close
N	Next Day Trade (Next Day Clearing)
O	Market Center Opening Trade
P	Prior Reference Price
Q	Market Center Official Open
R	Seller
T	Extended Hours Trade
U	Extended Hours Sold (Out Of Sequence)
V	Contingent Trade

Code	Value
X	Cross Trade
Z	Sold (Out Of Sequence)
4	Derivatively Priced
5	Market Center Reopening Trade
6	Market Center Closing Trade
7	Qualified Contingent Trade
8	Reserved
9	Corrected Consolidated Close Price as per Listing Market

UTDF

Below is a table containing condition codes from the UTP (Unlisted Trading Privileges) Plan:

For further details on this topic, please refer to page 43 of the UTP Specification.

Code	Value
@	Regular Sale
A	Acquisition
B	Bunched Trade
C	Cash Sale
D	Distribution
E	Placeholder
F	Intermarket Sweep
G	Bunched Sold Trade
H	Price Variation Trade
I	Odd Lot Trade`
K	Rule 155 Trade (AMEX)
L	Sold Last
M	Market Center Official Close

Code	Value
N	Next Day
O	Opening Prints
P	Prior Reference Price
Q	Market Center Official Open
R	Seller
S	Split Trade
T	Form T
U	Extended trading hours (Sold Out of Sequence)
V	Contingent Trade
W	Average Price Trade
X	Cross Trade
Y	Yellow Flag Regular Trade
Z	Sold (Out Of Sequence)
1	Stopped Stock (Regular Trade)
4	Derivatively Priced
5	Re-Opening Prints
6	Closing Prints
7	Qualified Contingent Trade (QCT)
8	Placeholder For 611 Exempt
9	Corrected Consolidated Close (per listing market)

Quote conditions

CQS

Below is a table containing codes that indicate specific conditions applicable to a quote under the CTA (Consolidated Tape Association) Plan:

For further details on this topic, please refer to Appendix G of the CQS Specification.

Code	Value
A	Slow Quote Offer Side
B	Slow Quote Bid Side
E	Slow Quote LRP Bid Side
F	Slow Quote LRP Offer Side
H	Slow Quote Bid And Offer Side
O	Opening Quote
R	Regular Market Maker Open
W	Slow Quote Set Slow List
C	Closing Quote
L	Market Maker Quotes Closed
U	Slow Quote LRP Bid And Offer
N	Non Firm Quote
4	On Demand Intra Day Auction

UQDF

Below is a table containing codes that represent specific conditions applicable to quotes under the UTP (Unlisted Trading Privileges) Plan:

For further details on this topic, please refer to the UQDF Specification.

Code	Value
A	Manual Ask Automated Bid
B	Manual Bid Automated Ask
F	Fast Trading
H	Manual Bid And Ask
I	Order Imbalance
L	Closed Quote
N	Non Firm Quote
O	Opening Quote Automated

Code	Value
R	Regular Two Sided Open
U	Manual Bid And Ask Non Firm
Y	No Offer No Bid One Sided Open
X	Order Influx
Z	No Open No Resume
4	On Demand Intra Day Auction

Trading API

Suggest Edits

Alpaca offers brokerage services for equities and crypto. Equity trading is commission free while crypto trading fees are tiered. Alpaca-py allows you to place orders and manage your positions on your Alpaca brokerage account.

Paper Trading

Alpaca offers a paper trading sandbox environment so you can test out the API or paper trade your strategy before you go live. The paper trading environment is free to use. You can learn more about paper trading here.

To use paper trading, you will need to set the paper parameter to True when instantiating the TradingClient. Make sure the keys you are providing correspond to a paper account.

Example

Python

```
from alpaca.trading.client import TradingClient

# paper=True enables paper trading
trading_client = TradingClient('api-key', 'secret-key', paper=True)
```

Retrieving Account Details

You can access details about your brokerage account like how much buying power you have, whether you've been flagged by as a pattern day trader, your total equity. You can learn more about the Account model here.

Example

Python
```python
from alpaca.trading.client import TradingClient

trading_client = TradingClient('api-key', 'secret-key')

account = trading_client.get_account()
```

Assets

The assets API serves a list of assets available on Alpaca for trading and data consumption. It is important to note that not all assets are tradable on Alpaca, and those assets will be marked with `tradable=False`. You can learn more about Assets, click here.

Getting All Assets

Retrieves a list of assets that matches the search parameters. If there is not any search parameters provided, a list of all available assets will be returned. Search parameters for assets are defined by the `GetAssetsRequest` model, which allows filtering by `AssetStatus`, `AssetClass`, and `AssetExchange`.
Example

Python
```python
from alpaca.trading.client import TradingClient
from alpaca.trading.requests import GetAssetsRequest
from alpaca.trading.enums import AssetClass

trading_client = TradingClient('api-key', 'secret-key')

# search for crypto assets
search_params = GetAssetsRequest(asset_class=AssetClass.CRYPTO)

assets = trading_client.get_all_assets(search_params)
```

Orders

The orders API allows you to submit orders and then manage those orders. You can customize your order with various order types, order time in forces or by creating multi-leg orders. You can learn more about order types at Alpaca here.

Creating an Order

To create on order on Alpaca-py you must use an `OrderRequest` object. There are different `OrderRequest` objects based on the type of order you want to make. For

market orders, there is `MarketOrderRequest`, limit orders have `LimitOrderRequest`, stop orders `StopOrderRequest`, and trailing stop orders have `TrailingStopOrderRequest`. Each order type have their own required parameters for a successful order.
😇

Hint

For stocks, the notional parameter can only be used with Market orders. For crypto, the notional parameter can be used with any order type.

Market Order

A market order is an order to buy or sell a stock at the best available price. Generally, this type of order will be executed immediately. However, the price at which a market order will be executed is not guaranteed.

Market orders allow the trade of fractional shares for stocks. Fractional shares must be denoted either with a non-integer `qty` value or with the use of the `notional` parameter. The `notional` parameter allows you to denote the amount you wish to trade in units of the quote currency. For example, instead of trading 1 share of SPY, we can trade $200 of SPY. `notional` orders are inherently fractional orders.

Example

Python

```python
from alpaca.trading.client import TradingClient
from alpaca.trading.requests import MarketOrderRequest
from alpaca.trading.enums import OrderSide, TimeInForce

trading_client = TradingClient('api-key', 'secret-key', paper=True)

# preparing orders
market_order_data = MarketOrderRequest(
                    symbol="SPY",
                    qty=0.023,
                    side=OrderSide.BUY,
                    time_in_force=TimeInForce.DAY
                    )

# Market order
market_order = trading_client.submit_order(
                order_data=market_order_data
               )
```

Limit Order

A limit order is an order to buy or sell a stock at a specific price or better. You can use the `LimitOrderRequest` model to prepare your order details.

Example

Python

```python
from alpaca.trading.client import TradingClient
from alpaca.trading.requests import LimitOrderRequest
from alpaca.trading.enums import OrderSide, TimeInForce

trading_client = TradingClient('api-key', 'secret-key', paper=True)


limit_order_data = LimitOrderRequest(
                    symbol="BTC/USD",
                    limit_price=17000,
                    notional=4000,
                    side=OrderSide.SELL,
                    time_in_force=TimeInForce.FOK
                   )

# Limit order
limit_order = trading_client.submit_order(
                order_data=limit_order_data
              )
```

Getting All Orders

We can attempt to cancel all open orders with this method. The method takes no parameters and returns a list of `CancelOrderResponse` objects. The cancellation of an order is not guaranteed. The `CancelOrderResponse` objects contain information about the cancel status of each attempted order cancellation.

Example

Python

```python
from alpaca.trading.client import TradingClient

trading_client = TradingClient('api-key', 'secret-key', paper=True)

# attempt to cancel all open orders
cancel_statuses = trading_client.cancel_orders()
```

Positions

The positions endpoints lets you track and manage open positions in your portfolio. You can learn more about the Position model here.

Getting All Positions

This method requires no parameters and returns all open positions in your portfolio. It will return a list of Position objects.

Example

Python
```python
from alpaca.trading.client import TradingClient

trading_client = TradingClient('api-key', 'secret-key')

trading_client.get_all_positions()
```

Close All Positions

This method closes all your open positions. If you set the `cancel_orders` parameter to True, the method will also cancel all open orders, preventing you from entering into a new position.
Example

Python
```python
from alpaca.trading.client import TradingClient

trading_client = TradingClient('api-key', 'secret-key')

# closes all position AND also cancels all open orders
trading_client.close_all_positions(cancel_orders=True)
```

Streaming Trade Updates

There is also a `TradingStream` websocket client which allows you to stream order updates. Whenever an order is submitted, filled, cancelled, etc, you will receive a response on the client.
Example

Python
```python
from alpaca.trading.stream import TradingStream

trading_stream = TradingStream('api-key', 'secret-key', paper=True)

async def update_handler(data):
    # trade updates will arrive in our async handler
    print(data)

# subscribe to trade updates and supply the handler as a parameter
trading_stream.subscribe_trade_updates(update_handler)

# start our websocket streaming
trading_stream.run()
```

Broker API

Suggest Edits

Getting Started with BrokerClient

In alpaca-py, all Broker API services are accessed through the BrokerClient. For each endpoint in the Broker API, there is a corresponding method within the client. To initialize a client, you will need to provide it your API keys which can be found on you Broker dashboard. If you wish to use your sandbox keys, you will need to set the sandbox parameter to `True` when initializing.
Learn more about `BrokerClient` in the Broker reference page.
Python

```
from alpaca.broker import BrokerClient

BROKER_API_KEY = "api-key"
BROKER_SECRET_KEY = "secret-key"

broker_client = BrokerClient(
                    api_key=Broker_API_KEY,
                    secret_key=BROKER_SECRET_KEY,
                    sandbox=True,
                )
```

Using Request Objects

In Alpaca-py, many methods will require instantiating and passing in a separate object for the request parameters. For example, the `BrokerClient::create_journal` method requires you to pass in a `CreateJournalRequest` object as a parameter. Once successfully instantiated, the `CreateJournalRequest` object will contain all the data required for a successful request to the API.

Accounts

The accounts API allows you to create and manage brokerage accounts on behalf of your users. To learn more about accounts on Broker API, click here.

Create an Account

You can create brokerage accounts on behalf of your users using the `BrokerClient::create_account` method. To create an account you need to first instantiate a `CreateAccountRequest` with all the relevant account

details. CreateAccountRequest requires contact, identity, disclosures, and agreements. There are also two additional fields which are optional: documents and trusted_contact.
First we will need to prepare our account data by organizing its constituent parts. Then we can pass those parts into the CreateAccountRequest model before submitting our request.
Python

```
from alpaca.broker.client import BrokerClient
from alpaca.broker.models import (
                        Contact,
                        Identity,
                        Disclosures,
                        Agreement
                    )
from alpaca.broker.requests import CreateAccountRequest
from alpaca.broker.enums import TaxIdType, FundingSource, AgreementType

broker_client = BrokerClient('api-key', 'secret-key')

# Contact
contact_data = Contact(
            email_address="cool_alpaca@example.com",
            phone_number="555-666-7788",
            street_address=["20 N San Mateo Dr"],
            city="San Mateo",
            state="CA",
            postal_code="94401",
            country="USA"
            )
# Identity
identity_data = Identity(
        given_name="John",
        middle_name="Smith",
        family_name="Doe",
        date_of_birth="1990-01-01",
        tax_id="666-55-4321",
        tax_id_type=TaxIdType.USA_SSN,
        country_of_citizenship="USA",
        country_of_birth="USA",
        country_of_tax_residence="USA",
        funding_source=[FundingSource.EMPLOYMENT_INCOME]
        )

# Disclosures
disclosure_data = Disclosures(
        is_control_person=False,
        is_affiliated_exchange_or_finra=False,
        is_politically_exposed=False,
        immediate_family_exposed=False,
        )

# Agreements
agreement_data = [
    Agreement(
      agreement=AgreementType.MARGIN,
      signed_at="2020-09-11T18:09:33Z",
      ip_address="185.13.21.99",
    ),
```

```
    Agreement(
      agreement=AgreementType.ACCOUNT,
      signed_at="2020-09-11T18:13:44Z",
      ip_address="185.13.21.99",
    ),
    Agreement(
      agreement=AgreementType.CUSTOMER,
      signed_at="2020-09-11T18:13:44Z",
      ip_address="185.13.21.99",
    ),
    Agreement(
      agreement=AgreementType.CRYPTO,
      signed_at="2020-09-11T18:13:44Z",
      ip_address="185.13.21.99",
    )
]

# ## CreateAccountRequest ## #
account_data = CreateAccountRequest(
                        contact=contact_data,
                        identity=identity_data,
                        disclosures=disclosure_data,
                        agreements=agreement_data
                        )

# Make a request to create a new brokerage account
account = broker_client.create_account(account_data)
```

List All Accounts

The BrokerClient::list_accounts method allows you to list all the brokerage accounts under your management. The method takes an optional parameter search_parameters which requires a ListAccountsRequest object. This parameter allows you to filter the list of accounts returned.

Python

```
from alpaca.broker.client import BrokerClient
from alpaca.broker.requests import ListAccountsRequest
from alpaca.broker.enums import AccountEntities

broker_client = BrokerClient('api-key', 'secret-key')

# search for accounts created after January 30th 2022.
#Response should contain Contact and Identity fields for each account.
filter = ListAccountsRequest(
                    created_after=datetime.datetime.strptime("2022-01-30", "%Y-%m-
%d"),
                    entities=[AccountEntities.CONTACT, AccountEntities.IDENTITY]
                    )

accounts = broker_client.list_accounts(search_parameters=filter)
```

Funding

The funding API allows you to create Bank/ACH connections and transfer funds in and out of accounts. To learn more about funding on Broker API, click here.

Create an ACH Relationship

Before an account can be funded, it needs to have an external account connection established. There are two types of connections that be created: ACH relationships and bank relationships. ACH Relationships can be created using routing and account numbers, or via Plaid.

To use Plaid, you will require a processor_token provided by Plaid specifically for Alpaca. View this article to learn more

In this example we will use routing and account numbers to establish an ACH relationship.

Python

```python
from alpaca.broker.client import BrokerClient
from alpaca.broker.requests import CreateACHRelationshipRequest
from alpaca.broker.enums import BankAccountType

broker_client = BrokerClient('api-key', 'secret-key')

account_id = "c8f1ef5d-edc0-4f23-9ee4-378f19cb92a4"

ach_data = CreateACHRelationshipRequest(
                    account_owner_name="John Doe",
                    bank_account_type=BankAccountType.CHECKING,
                    bank_account_number="123456789abc",
                    bank_routing_number="121000358",
                )

ach_relationship = broker_client.create_ach_relationship_for_account(
                    account_id=account_id,
                    ach_data=ach_data
                )
```

Create a Transfer

After a bank or ACH relationship has been established for an account, transfers can be made. There are two types of transfers: incoming (deposits) or outgoing (withdrawals). Transfers based on ACH relationships must use CreateACHTransferRequest and bank relationships must use CreateBankTransferRequest.

Python

```python
from alpaca.broker.client import BrokerClient
from alpaca.broker.requests import CreateACHTransferRequest
from alpaca.broker.enums import TransferDirection, TransferTiming

broker_client = BrokerClient('api-key', 'secret-key')
```

```
account_id = "c8f1ef5d-edc0-4f23-9ee4-378f19cb92a4"

transfer_data = CreateACHTransferRequest(
                    amount=1000,
                    direction=TransferDirection.INCOMING,
                    timing=TransferTiming.IMMEDIATE,
                    relationship_id="0f08c6bc-8e9f-463d-a73f-fd047fdb5e94"
                )
transfer = broker_client.create_transfer_for_account(
                account_id=account_id,
                transfer_data=transfer_data
            )
```

Journals

The journals API allows you to transfer cash and securities between accounts under your management. To learn more about the journals API, click here.

Create a Journal

A journal is made between two accounts. For every journal request, assets will leave from_account and into to_account. There are two types of journals: cash journals and security journals. Cash journals move the account currency between accounts. Security journals move stocks between accounts.

Python

```
from alpaca.broker.client import BrokerClient
from alpaca.broker.requests import CreateJournalRequest
from alpaca.broker.enums import JournalEntryType

broker_client = BrokerClient('api-key', 'secret-key')

journal_data = CreateJournalRequest(
                    from_account="c8f1ef5d-edc0-4f23-9ee4-378f19cb92a4",
                    entry_type=JournalEntryType.CASH,
                    to_account="0f08c6bc-8e9f-463d-a73f-fd047fdb5e94",
                    amount=50
                )

journal = broker_client.create_journal(journal_data=journal_data)
```

Create a Batch Journal

A batch journal lets you journal from one account into many accounts at the same time.

Python

```
from alpaca.broker.client import BrokerClient
from alpaca.broker.requests import CreateBatchJournalRequest,
BatchJournalRequestEntry
```

```python
from alpaca.broker.enums import JournalEntryType

broker_client = BrokerClient('api-key', 'secret-key')

# Receiving accounts
batch_entries = [
    BatchJournalRequestEntry(
            to_account="d7017fd9-60dd-425b-a09a-63ff59368b62",
            amount=50,
    ),
    BatchJournalRequestEntry(
            to_account="94fa473d-9a92-40cd-908c-25da9fba1e65",
            amount=100,
    ),
    BatchJournalRequestEntry(
            to_account="399f85f1-cbbd-4eaa-a934-70027fb5c1de",
            amount=700,
    ),
]

batch_journal_data = CreateBatchJournalRequest(
                    entry_type=JournalEntryType.CASH,
                    from_account="8f8c8cee-2591-4f83-be12-82c659b5e748",
                    entries=batch_entries
                )

batch_journal = broker_client.create_batch_journal(batch_data=batch_journal_data)
```

Trading

The Broker trading API allows you to place orders and manage positions on behalf of your users. To learn more about trading on Broker API, click here.

Broker API vs Trading API

Keep in mind, all models necessary for trading on Broker API live within the alpaca.broker and not alpaca.trading. Although the trading models in alpaca.broker and alpaca.trading have the same name, they are different.

Create an Order

To create an order on Alpaca-py you must use an OrderRequest object. There are different OrderRequest objects based on the type of order you want to make. For market orders, there is MarketOrderRequest, limit orders have LimitOrderRequest, stop orders StopOrderRequest, and trailing stop orders have TrailingStopOrderRequest. Each order type have their own required parameters for a successful order.
Python

```python
from alpaca.broker.client import BrokerClient
from alpaca.broker.requests import MarketOrderRequest, LimitOrderRequest
from alpaca.trading.enums import OrderSide, TimeInForce

broker_client = BrokerClient('api-key', 'secret-key')

# account to make order for
account_id = "c8f1ef5d-edc0-4f23-9ee4-378f19cb92a4"

# preparing orders
market_order_data = MarketOrderRequest(
                    symbol="BTCUSD",
                    qty=1,
                    side=OrderSide.BUY,
                    time_in_force=TimeInForce.GTC,
                    commission=1
            )

limit_order_data = LimitOrderRequest(
                    symbol="SPY",
                    limit_price=300,
                    qty=10,
                    side=OrderSide.SELL,
                    time_in_force=TimeInForce.FOK,
                    commission=1
              )

# Market order
market_order = broker_client.submit_order_for_account(
                account_id=account_id,
                order_data=market_order_data
                )

# Limit order
limit_order = broker_client.submit_order_for_account(
                account_id=account_id,
                order_data=limit_order_data
               )
```

Get All Positions

You can retrieve all open positions for a specific account using only the account_id. This will return a list of Position objects.

Python

```python
from alpaca.broker import BrokerClient

broker_client = BrokerClient('api-key', 'secret-key')

# account to get positions for
account_id = "c8f1ef5d-edc0-4f23-9ee4-378f19cb92a4"

positions = broker_client.get_all_positions_for_account(account_id=account_id)
```