

DOCS

- [Introduction](#)
- [CLOB API](#)
 - [Introduction](#)
 - [System](#)
 - [API](#)
 - [Security](#)
 - [Fees](#)
 - [Additional Resources](#)
 - [Deployments](#)
 - [Status](#)
 - [Clients](#)
 - [Endpoints](#)
 - [Authentication](#)
 - [Orders](#)
 - [Trades](#)
 - [Markets](#)
 - [Prices and Books](#)
 - [Websocket API](#)
 - [Timeseries Data](#)
- [Gamma Markets API](#)
 - [Overview](#)
 - [Endpoint](#)
 - [Market Organization](#)
 - [Markets](#)
 - [Single Market](#)
 - [Events](#)
 - [Single Event](#)
- [Subgraph](#)
 - [Overview](#)
 - [Source](#)
 - [Hosted Version](#)
- [Resolution](#)
 - [UMA](#)
 - [Pyth](#)
- [Rewards](#)
 - [Liquidity Rewards Program](#)
 - [Leaderboard Competitions](#)
- [Conditional Tokens Framework](#)
 - [Overview](#)
 - [Split](#)
 - [Merge](#)
 - [Redeem](#)
 - [Deployment](#)
 - [Resources](#)
- [FPMs](#)
 - [Overview](#)
 - [Add Liquidity](#)
 - [Remove Liquidity](#)

- [Buy](#)
 - [Sell](#)
 - [Deployments](#)
 - [Resources](#)
- [Proxy Wallets](#)
 - [Overview](#)
 - [Deployments](#)
- [Negative Risk](#)
 - [Overview](#)
 - [Augmented Negative Risk](#)
- [Bug Bounty](#)
- [Frequently Asked Questions](#)
 - [CLOB Client Questions](#)
 - [Onchain questions](#)

Introduction

Welcome to Polymarket's docs! Here developers can find all the information they need for interacting with Polymarket. This includes documentation on market discovery, resolution, trading etc. Whether you are an academic researcher a market maker or an indy developer, this documentation should provide you what you need to get started. All the code you find linked here and on our GitHub is open source and free to use. If you have any questions please join our discord and direct your questions to the #devs channel.

Access status

Please regularly check the access status of your account with the `ban-status/` endpoint. If the endpoint returns a value of `true` for `cert_required`, proof of residence is required and failure to provide it within 14 days will result in a close only status. To certify you are not breaching ToS please send an email to ops@polymarket.com with your address, form of id (passport, license, or other) and proof of residence (recent utility bill, bank bill, phone bill). You will also be asked to sign and return a non-US certification in subsequent communications. Once complete, within 24 hours the cert required status should be market to false.

CLOB API

Introduction

Welcome to the Polymarket Order Book API! In this documentation you will find overviews, explanations, examples and annotations that aim to make interacting with the order book a breeze. In this section we will provide a general overview of the Polymarket Order Book and the purpose of the API before diving deep into the API and clients in following sections.

SYSTEM

Polymarket's Order Book, also referred to as the "CLOB" (Central Limit Order Book) or "BLOB" (Binary Limit Order Book), is hybrid-decentralized wherein there is an operator that provides off-chain matching/ordering services while settlement/execution happens on-chain, non-custodially according to instructions provided by users in the form of signed order messages. This decentralized exchange model provides users with a powerful, non-custodial exchange experience.

Underlying the exchange system is a custom Exchange contract that facilitates atomic swaps (settlement) between binary Outcome Tokens (both CTF ERC1155 assets and ERC20 PToken assets) and a collateral asset (ERC20) according to signed limit orders. The Exchange contract is purpose built for binary markets (instruments that allow collateral to be split into positions and conversely positions to be merged into collateral with the two positions ultimately being settled to a price equal to 1). This allows "unification" of order books such that orders for a position and its complement can be matched. Explicitly, the Exchange allows for matching operations that include a mint/merge operation which allows orders for complementary outcome tokens to be crossed.

Orders are represented as signed typed structured data (EIP712). When orders are matched, one side is considered the maker and the other side is considered the taker. The relationship is always either one to one or many to one (maker to taker) and any price improvement is captured by the taker. The Operator is responsible for matching, ordering, and submitting matched trades to the underlying blockchain network for execution. As such, order placement and cancellation can happen immediately off-chain while only the settlement action must occur on-chain.

API

The Polymarket Order Book API is a set of endpoints that allow market makers, traders, and other Polymarket users to programmatically create and manage orders for markets via access to the API provided by the operator.

Orders for any amount can be created and listed, or fetched and read from the order book for a given market. The API also provides data on all available markets, market prices, and order history through REST and WSS endpoints.

SECURITY

Polymarket's Exchange contract has been audited by Chainsecurity you can find the audit report [here](#).

The operator has no special privileges outside of ordering, this means that the only actions you must trust them with is enforcing correct ordering, not censoring and removing cancellations (orders can also be cancelled on-chain if operator is not trusted). If the operator is not doing any of these activities fairly a user can simply stop interacting with the operator. The operator is never able to set prices for users, or execute any trade on the user's behalf outside of the signed limit orders the user creates.

FEES

SCHEDULE

Subject to change

Volume Level	Maker Fee Base Rate (bps)	Taker Fee Base Rate (bps)
>0 USDC	0	0

OVERVIEW

Fees are levied in the output asset (proceeds). Fees for binary options with a complementary relationship (ie $A + A' = C$) must be symmetric to preserve market integrity. Symmetric means that someone selling 100 shares of A @ \$0.99 should pay the same fee value as someone buying 100 A' @ \$0.01. An intuition for this requires understanding that minting/merging a complementary token set for collateral can happen at any time. Fees are thus implemented in the following manner.

If buying (ie receiving A or A'), the fee is levied on the proceed tokens. If selling (ie receiving C), the fee is levied on the proceed collateral. The base fee rate (baseFeeRate) is signed into the order struct. The base fee rate corresponds to % the fee rate paid by traders when the price of the two tokens is equal (ie 0.50 and 0.50). Moving away from a centered price, the following formulas are used to calculate the fees making sure to maintain symmetry.

Case 1: If selling outcome tokens (base) for collateral (quote):

$$\text{feeQuote} = \text{baseRate} * \min(\text{price}, 1 - \text{price}) * \text{size}$$

Case 2: If buying outcome tokens (base) with collateral (quote):

$$\text{feeBase} = \text{baseRate} * \min(\text{price}, 1 - \text{price}) * \frac{\text{size}}{\text{price}}$$

ADDITIONAL RESOURCES

- Exchange contract source code
- Exchange contract documentation

Deployments

The Exchange contract is deployed at the following addresses:

Network	Address
Mumbai:	0x4bFb41d5B3570DeFd03C39a9A4D8dE6Bd8B8982E
Polygon:	0x4bFb41d5B3570DeFd03C39a9A4D8dE6Bd8B8982E

Status

<https://status-clob.polymarket.com/>

Clients

Installation

```
npm i -s @polymarket/clob-client
```

```
yarn add @polymarket/clob-client
```

Polymarket has implemented reference clients that allow programmatic use of the API below:

- clob-client (Typescript)
- py-clob-client (Python)

Initialization

```
import { ClobClient } from "polymarket/clob-client";
import { SignatureType } from "@polymarket/order-utils";

// Initialization of a client that trades directly from an EOA
const clobClient = new ClobClient(
  host as string,
  (await wallet.getChainId()) as number,
  wallet as ethers.Wallet | ethers.providers.JsonRpcSigner
);

// Initialization of a client using a Polymarket Proxy associated with an Email/Magic account
const clobClient = new ClobClient(
  host as string,
  (await wallet.getChainId()) as number,
  wallet as ethers.Wallet | ethers.providers.JsonRpcSigner,
  undefined, // creds
```

```
SignatureType.POLY_PROXY,  
"YOUR_POLYMARKET_PROXY_ADDRESS"  
);  
  
// Initialization of a client using a Polymarket Proxy Wallet associated with a Browser Wallet(Metamask, Coinbase Wallet)  
const clobClient = new ClobClient(  
  host as string,  
  (await wallet.getChainId()) as number,  
  wallet as ethers.Wallet | ethers.providers.JsonRpcSigner,  
  undefined, // creds  
  SignatureType.POLY_GNOSIS_SAFE,  
  "YOUR_POLYMARKET_PROXY_ADDRESS"  
);
```

ORDER UTILS

Polymarket has implemented utility libraries to programmatically sign and generate orders:

- clob-order-utils (Typescript)
- python-order-utils (Python)
- go-order-utils (Golang)

Endpoints

REST

Used for all CLOB REST endpoints, denoted {clob-endpoint}.

https://clob.polymarket.com/

WEBSOCKET

Used for all CLOB WSS endpoints, denoted {wss-channel}.

wss://ws-subscriptions-clob.polymarket.com/ws/

Authentication

There are two levels of authentication to be considered when using Polymarket's CLOB. All signing can be handled directly by the client libraries.

L1: PRIVATE KEY AUTHENTICATION

The highest level of authentication is via an account's Polygon private key. The private key remains in control of a user's funds and as such all trading is non-custodial. The operator *never* has control over users' funds.

Private key authentication is required for the following operations:

- Placing an order (for signing the order)
- Creating or revoking API keys

L1 HEADER

Header	Required?	Description
POLY_ADDRESS	yes	Polygon address

Header	Required?	Description
POLY_SIGNATURE	yes	CLOB EIP 712 signature
POLY_TIMESTAMP	yes	current UNIX timestamp
POLY_NONCE	yes	Nonce. Default 0

The POLY_SIGNATURE is generated by signing the following EIP712 struct:

```
const domain = {
  name: "ClobAuthDomain",
  version: "1",
  chainId: chainId, // Polygon ChainID 137
};

const types = {
  ClobAuth: [
    { name: "address", type: "address" },
    { name: "timestamp", type: "string" },
    { name: "nonce", type: "uint256" },
    { name: "message", type: "string" },
  ],
};

const value = {
  address: signingAddress, // the Signing address
  timestamp: ts, // The CLOB API server timestamp
  nonce: nonce, // The nonce used
  message: "This message attests that I control the given wallet", // A static message indicating that the user controls the wallet
};

const sig = await signer._signTypedData(domain, types, value);
```

Implementations exist in the Typescript and Python clients

L2: API KEY AUTHENTICATION

The next level of authentication consists of the API key, secret and passphrase which are used solely to authenticate API requests made to Polymarket's CLOB. This includes operations such as posting/canceling orders or retrieving an account's orders and fills.

When a user on-boards via `POST /auth/api-key`, the server will use the signature as a seed to deterministically generate credentials. An API credential includes three fields:

key: UUID identifying the credentials.

secret: Secret string used to generate HMACs, not sent with requests.

passphrase: Secret string sent with each request, used to encrypt/decrypt the secret in our DB, and never stored in our DB.

All requests which are not signed by a private key (`/auth/api-key`) and which are made to private endpoints require an API key signature.

L2 HEADER

Header	Required?	Description
POLY_ADDRESS	yes	Polygon address
POLY_SIGNATURE	yes	HMAC signature for request
POLY_TIMESTAMP	yes	request UNIX timestamp

Header	Required?	Description
POLY_API_KEY	yes	Polymarket API key
POLY_PASSPHRASE	yes	Polymarket API key passphrase

CREATE API KEY

i This endpoint requires an L1 Header.

Create new API key credentials for a user.

HTTP REQUEST

POST {clob-endpoint}/auth/api-key

```
const creds = await clobClient.createApiKey(); // nonce defaults to 0
```

The above request returns a JSON structured like this:

```
{
  "apiKey": "xxxxxxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx",
  "secret": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx=",
  "passphrase": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
}
```

DERIVE API KEY

i This endpoint required a L1 Header.

Derive an *existing* API key for an address and nonce.

HTTP REQUEST

GET {clob-endpoint}/auth/derive-api-key

```
const creds = await clobClient.deriveApiKey(); // nonce defaults to 0
```

The above request returns JSON structured like this:

```
{
  "apiKey": "xxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "secret": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx=",
  "passphrase": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
}
```

GET API KEYS

i This endpoint required a L2 Header.

Get all api keys associated with a Polygon address.

HTTP REQUEST


```
GET {clob-endpoint}/auth/api-keys
```

```
const apiKeys = await clobClient.getApiKeys();
```

The above command returns a JSON structured like this:

```
{
  "apiKeys": [ "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx", ... ]
}
```

DELETE API KEY

 This endpoint required a L2 Header.

Deletes API used to authenticate the request.

HTTP REQUEST

```
DELETE {clob-endpoint}/auth/api-key
```

```
const resp = await clobClient.deleteApiKey();
console.log(resp);
```

If successful, the above command returns an `"OK"` string value:

```
"OK"
```

ACCESS STATUS

This endpoint returns the value of `cert_required` by signer address. Please, check this regularly and if it returns a value of true for `cert_required`, refer to this section

HTTP REQUEST

```
GET '{clob-endpoint}/auth/ban-status/cert-required?address={signer address}'
```

RESPONSE

```
{
  "cert_required": false | true
}
```

Orders

OVERVIEW

All orders are expressed as limit orders (can be marketable). The underlying order primitive must be in the form expected and executable by the on-chain binary limit order protocol contract. Preparing such an order is quite involved (structuring, hashing, signing), thus Polymarket suggests using the open source typescript, python and go lang libraries.

ALLOWANCES

To place an order, allowances must be set by the funder address for the specified `maker` asset for the Exchange contract. When buying, this means the funder must have set a USDC allowance greater than or equal to the spending amount. When selling, the funder must have set an allowance for the conditional token that is greater than or equal to the selling amount. This allows the Exchange contract to execute settlement according to the signed order instructions created by a user and matched by the operator.

SIGNATURE TYPES

Polymarket's CLOB supports 3 signature types. Orders must identify what signature type they use. The available typescript and python clients abstract the complexity of signing and preparing orders with the following signature types by allowing a funder address and signer type to be specified on initialization. The supported signature types are:

Type	ID	Description
EOA	0	EIP712 signature signed by an EOA
POLY_PROXY	1	EIP712 signatures signed by a signer associated with funding Polymarket proxy wallet
POLY_GNOSIS_SAFE	2	EIP712 signatures signed by a signer associated with funding Polymarket gnosis safe wallet


VALIDITY CHECKS

Orders are continually monitored to make sure they remain valid. Specifically, this includes continually tracking underlying balances, allowances and on-chain order cancellations. Any maker that is caught intentionally abusing these checks (which are essentially real time) will be blacklisted.

Additionally, there are rails on order placement in a market. Specifically, you can only place orders that sum to less than or equal to your available balance for each market. For example if you have 500 USDC in your funding wallet, you can place one order to buy 1000 YES in marketA @ \$.50, then any additional buy orders to that market will be rejected since your entire balance is reserved for the first (and only) buy order. More explicitly the max size you can place for an order is:

$$\text{maxOrderSize} = \text{underlyingAssetBalance} - \sum (\text{orderSize} - \text{orderFillAmount})$$

CREATE AND PLACE AN ORDER

 This endpoint required a L2 Header.

Create and place an order using the Polymarket CLOB API clients. All orders are represented as "limit" orders, but "market" orders are enabled by allowing marketable limit orders, which are executed on input at the best price available. Thus to place a market order, simply ensure your price is marketable against current resting limit orders.

HTTP REQUEST

POST `{clob-endpoint}/order`

```
// GTC Order example
//
import { Side, OrderType } from "@polymarket/clob-client";

async function main() {
  // Create a buy order for 100 YES for 0.50c
  // YES: 71321045679252212594626385532706912750332728571942532289631379312455583992563
  const order = await clobClient.createOrder({
    tokenId:
      "71321045679252212594626385532706912750332728571942532289631379312455583992563",
    price: 0.5,
    side: Side.BUY,
```

```

    size: 100,
    feeRateBps: 100,
    nonce: 1,
  });
  console.log("Created Order", order);

  // Send it to the server

  // GTC Order
  const resp = await clobClient.postOrder(order, OrderType.GTC);
  console.log(resp);
}

main();

```

```

// GTD Order example
//
import { Side, OrderType } from "@polymarket/clob-client";

async function main() {
  // Create a buy order for 100 YES for 0.50c that expires in 1 minute
  // YES: 71321045679252212594626385532706912750332728571942532289631379312455583992563

  // There is a 1 minute of security threshold for the expiration field.
  // If we need the order to expire in 30 seconds the correct expiration value is:
  // now + 1 miute + 30 seconds
  const oneMinute = 60 * 1000;
  const seconds = 30 * 1000;
  const expiration = parseInt(
    (new Date().getTime() + oneMinute + seconds) / 1000).toString()
  );

  const order = await clobClient.createOrder({
    tokenID:
      "71321045679252212594626385532706912750332728571942532289631379312455583992563",
    price: 0.5,
    side: Side.BUY,
    size: 100,
    feeRateBps: 100,
    nonce: 1,
    // There is a 1 minute of security threshold for the expiration field.
    // If we need the order to expire in 30 seconds the correct expiration value is:
    // now + 1 miute + 30 seconds
    expiration: expiration,
  });
  console.log("Created Order", order);

  // Send it to the server

  // GTD Order
  const resp = await clobClient.postOrder(order, OrderType.GTD);
  console.log(resp);
}

main();

```

```

// FOK Order example
//
import { Side, OrderType } from "@polymarket/clob-client";

async function main() {
  // Create a buy order for 100 YES for 0.50c that expires in 1 minute
  // YES: 71321045679252212594626385532706912750332728571942532289631379312455583992563

  const oneMinute = parseInt(

```

```

    ((new Date().getTime() + 60 * 1000 + 10 * 1000) / 1000).toString()
  );

  const marketOrder = await clobClient.createMarketBuyOrder({
    tokenID:
      "71321045679252212594626385532706912750332728571942532289631379312455583992563",
    amount: 100,
    feeRateBps: 0,
    nonce: 0,
    price: 0.5,
  });
  console.log("Created Order", order);

  // Send it to the server

  // FOK Order
  const resp = await clobClient.postOrder(order, OrderType.FOK);
  console.log(resp);
}

main();

```

REQUEST PAYLOAD PARAMETERS

Name	Required	Type	Description
order	yes	Order	signed order object
owner	yes	string	api key of oder owner
orderType	yes	string	order type ("FOK", "GTC", "GTD")

An `Order` object is of the form:

Name	Required	Type	Description
salt	yes	integer	random salt used to create unique order
maker	yes	string	maker address (funder)
signer	yes	string	signing address
taker	yes	string	taker address (operator)
tokenId	yes	string	ERC1155 token ID of conditional token being traded
makerAmount	yes	string	maximum amount maker is willing to spend
takerAmount	yes	string	minimum amount taker must pay the maker in return
expiration	yes	string	unix expiration timestamp
nonce	yes	string	maker's Exchange nonce the order is associated with
feeRateBps	yes	string	fee rate in basis points as required by the operator
side	yes	string	buy or sell enum index
signatureType	yes	integer	signature type enum index

Name	Required	Type	Description
signature	yes	string	hex encoded signature

ORDER TYPES:

- **FOK**: A 'Fill-Or-Kill' order is an market order to buy shares that must be executed immediately in its entirety; otherwise, the entire order will be cancelled.
- **GTC**: A 'Good-Til-Cancelled' order is a limit order that is active until it is fulfilled or cancelled.
- **GTD**: A 'Good-Til-Day' order is a type of order that is active until its specified date (UTC seconds timestamp), unless it has already been fulfilled or cancelled. There is a security threshold of one minute: If the order needs to expire in 30 seconds the correct expiration value is: now + 1 miute + 30 seconds

RESPONSE FORMAT

Name	Type	Description
success	boolean	boolean indicating server-side error (<code>if success == false</code> -> server-side error)
errorMsg	string	error message in case of unsuccessful placement (in case <code>success == true && errorMsg != ''</code> -> client-side error, the reason is in <code>errorMsg</code>)
orderId	string	id of order
transactionsHashes	string[]	hash of settlement transaction if order was marketable and triggered a match action
status	string	order status

INSERT ERRORS/MESSAGES

If the `errorMsg` field of the response object from placement is not an empty string the order was not able to be immediately placed. This might be because of a delay or because of a failure. If the `success` is not `true` then there was an issue placing the order. The following errors/messages are possible.

Error	Failure?	Message	Description
INVALID_ORDER_MIN_TICK_SIZE	yes	order is invalid. Price () breaks minimum tick size rule:	order price isn't accurate to correct tick sizing
INVALID_ORDER_MIN_SIZE	yes	order is invalid. Size () lower than the minimum:	order size must meet min size threshold requirement
INVALID_ORDER_DUPLICATED	yes	order is invalid. Duplicated.	same order has already been placed, can't be placed again
INVALID_ORDER_NOT_ENOUGH_BALANCE	yes	not enough balance / allowance	funder address doesn't have sufficient balance or allowance for order
INVALID_ORDER_EXPIRATION	yes	invalid expiration	expiration field expresses a time before now
INSERT_ORDER_ERROR	yes	could not insert order	system error while inserting order
EXECUTION_ERROR	yes	could not run the execution	system error while attempting to execute trade
ORDER_DELAYED	no	order match delayed due to market conditions	order placement delayed

Error	Failure?	Message	Description
DELAYING_ORDER_ERROR	yes	error delaying the order	system error while delaying order
FOK_ORDER_NOT_FILLED_ERROR	yes	order couldn't be fully filled. FOK orders are fully filled or killed.	FOK order not fully filled so can't be placed
MARKET_NOT_READY	no	the market is not yet ready to process new orders	system not accepting orders for market yet

INSERT STATUSES

When placing an order, a status field is included. This status field provides additional information regarding the order's state as a result of the placement. Possible values include:

Status	Description
live	order placed and live
matched	order matched (marketable)
delayed	order marketable, but subject to matching delay
unmatched	order marketable, but failure delaying, placement not successful

GET ORDER

 This endpoint required a L2 Header.

Get single order by id.

HTTP REQUEST

GET {clob-endpoint}/data/order/{order hash}

```
async function main() {
  const order = await clobClient.getOrder(
    "0xb816482a5187a3d3db49cbaf6fe3ddf24f53e6c712b5a4bf5e01d0ec7b11dabc"
  );
  console.log(order);
}

main();
```

REQUEST PARAMETERS

Name	Required	Type	Description
id	no	string	id of order to get information about

RESPONSE FORMAT

Name	Type	Description
null	OpenOrder	order if it is exists

An `OpenOrder` object is of the form:

Name	Type	Description
associate_trades	string[]	any Trade id the order has been partially included in
id	string	order id
status	string	order current status
market	string	market id (condition id)
original_size	string	original order size at placement
outcome	string	human readable outcome the order is for
maker_address	string	maker address (funder)
owner	string	api key
price	string	price
side	string	buy or sell
size_matched	string	size of order that has been matched/filled
asset_id	string	token id
expiration	string	unix timestamp when the order expired, 0 if it does not expire
type	string	order type (GTC, FOK, GTD)
created_at	string	unix timestamp when the order was created

CHECK IF AN ORDER IS SCORING

 This endpoint required a L2 Header.

Returns a boolean value where it is indicated if an order is scoring or not.

HTTP REQUEST

`GET {clob-endpoint}/order-scoring?order_id={...}`

```
async function main() {
  const scoring = await clobClient.isOrderScoring({
    orderId: "0x...",
  });
  console.log(scoring);
}

main();
```

REQUEST PARAMETERS

Name	Required	Type	Description
orderId	yes	string	id of order to get information about

RESPONSE FORMAT

Name	Type	Description
null	OrderScoring	order scoring data

An `OrderScoring` object is of the form:

Name	Type	Description
scoring	boolean	indicates if the order is scoring or not

CHECK IF SOME ORDERS ARE SCORING

 This endpoint required a L2 Header.

Returns a a dictionary with boolean value where it is indicated if an order is scoring or not.

HTTP REQUEST

POST `{clob-endpoint}/orders-scoring`

```
async function main() {
  const scoring = await clobClient.areOrdersScoring({
    orderIds: ["0x...",],
  });
  console.log(scoring);
}

main();
```

REQUEST PARAMETERS

Name	Required	Type	Description
orderIds	yes	string[]	ids of the orders to get information about

RESPONSE FORMAT

Name	Type	Description
null	OrdersScoring	orders scoring data

An `OrdersScoring` object is a dictionary that indicates order by order if it scores:

```
{
  "0x0": true,
  "0x1": false
}
```

GET ACTIVE ORDERS

i This endpoint required a L2 Header.

Get active order(s) for a specific market.

HTTP REQUEST

GET {clob-endpoint}/data/orders

```
async function main() {
  const resp = await clobClient.getOpenOrders({
    market:
      "0xbd31dc8a20211944f6b70f31557f1001557b59905b7738480ca09bd4532f84af",
  });
  console.log(resp);
  console.log(`Done!`);
}

main();
```

REQUEST PARAMETERS

Name	Required	Type	Description
id	no	string	id of order to get information about
market	no	string	condition id of market
asset_id	no	string	id of the asset/token

RESPONSE FORMAT

Name	Type	Description
null	OpenOrder[]	list of open orders filtered by query parameters

CANCEL AN ORDER

i This endpoint required a L2 Header.

Cancel an order.

HTTP REQUEST

DELETE {clob-endpoint}/order

```
async function main() {
  // Send it to the server
  const resp = await clobClient.cancelOrder({
    orderID:
      "0x38a73eed1e6d177545e9ab027abddfb7e08dbe975fa777123b1752d203d6ac88",
  });
  console.log(resp);
  console.log(`Done!`);
}

main();
```

REQUEST PAYLOAD PARAMETERS

Name	Required	Type	Description
orderId	yes	string	ID of order to cancel

RESPONSE FORMAT

Name	Type	Description
canceled	string[]	list of canceled orders
not_canceled	{ }	a order id - reason map that explains why that order couldn't be canceled

CANCEL ORDERS

 This endpoint required a L2 Header.

Cancel many orders.

HTTP REQUEST

DELETE {clob-endpoint}/orders

```
async function main() {
  // Send it to the server
  const resp = await clobClient.cancelOrders([
    "0x38a73eed1e6d177545e9ab027abddf7e08dbe975fa777123b1752d203d6ac88",
    "0xaaaa...",
  ]);
  console.log(resp);
  console.log(`Done!`);
}
main();
```

REQUEST PAYLOAD PARAMETERS

Name	Required	Type	Description
null	yes	string[]	IDs of the orders to cancel

RESPONSE FORMAT

Name	Type	Description
canceled	string[]	list of canceled orders
not_canceled	{ }	a order id - reason map that explains why that order couldn't be canceled

CANCEL ALL ORDERS

 This endpoint required a L2 Header.

Cancel all open orders posted by a user.

HTTP REQUEST

DELETE {clob-endpoint}/cancel-all


```
async function main() {
  const resp = await clobClient.cancelAll();
  console.log(resp);
  console.log(`Done!`);
}

main();
```

RESPONSE FORMAT

Name	Type	Description
canceled	string[]	list of canceled orders
not_canceled	{ }	a order id - reason map that explains why that order couldn't be canceled

CANCEL ORDERS FROM MARKET

 This endpoint required a L2 Header.

Cancel orders from market.

HTTP REQUEST

DELETE {clob-endpoint}/cancel-market-orders

```
async function main() {
  // Send it to the server
  const resp = await clobClient.cancelMarketOrders({
    market:
      "0xbd31dc8a20211944f6b70f31557f1001557b59905b7738480ca09bd4532f84af",
    asset_id:
      "52114319501245915516055106046884209969926127482827954674443846427813813222426",
  });
  console.log(resp);
  console.log(`Done!`);
}

main();
```

REQUEST PAYLOAD PARAMETERS

Name	Required	Type	Description
market	no	string	condition id of the market
asset_id	no	string	id of the asset/token

RESPONSE FORMAT

Name	Type	Description
canceled	string[]	list of canceled orders
not_canceled	{ }	a order id - reason map that explains why that order couldn't be canceled

Trades

OVERVIEW

All historical trades can be fetched via the Polymarket CLOB REST API. A trade is initiated by a "taker" who creates a marketable limit order. This limit order can be matched against one or more resting limit orders on the associated book. A trade can be in various states as described below. Note in some cases (due to gas limitations) the execution of a "trade" must be broken into multiple transactions in which case separate trade entities will be returned. To make these trade entities associable, there is a `bucket_index` field, a `market_order` field and a `match_time` field. Trades that have been broken into multiple trade objects can be reconciled by combining trade objects with the same `market_order`, `match_time` and incrementing `bucket_index`s into a top level "trade" client side.

STATUSES

Status	Terminal?	Description
MATCHED	no	trade has been matched and sent to the executor service by the operator, the executor service submits the trade as a transaction to the Exchange contract
MINED	no	trade is observed to be mined into the chain, no finality threshold established
CONFIRMED	yes	trade has achieved strong probabilistic finality and was successful
RETRYING	no	trade transaction has failed (revert or reorg) and is being retried/resubmitted by the operator
FAILED	yes	trade has failed and is not being retried

GET TRADES

 This endpoint required a L2 Header.

Get trades for the authenticated user based on the provided filters.

HTTP REQUEST

GET {clob-endpoint}/data/trades

```
async function main() {
  const trades = await clobClient.getTrades({
    market:
      "0xbd31dc8a20211944f6b70f31557f1001557b59905b7738480ca09bd4532f84af",
    maker_address: await wallet.getAddress(),
  });
  console.log(`trades: `);
  console.log(trades);
}

main();
```

REQUEST PARAMETERS

Name	Required	Type	Description
id	no	string	id of trade to fetch

Name	Required	Type	Description
taker	taker or maker	string	address to get trades for where it is included as a taker
maker	maker or taker	string	address to get trades for where it is included as a maker
market	no	string	market for which to get trades for (condition ID)
before	no	string	unix timestamp representing the cutoff up to which trades that happened before then can be included
after	no	string	unix timestamp representing the cutoff for which trades that happened after can be included

RESPONSE FORMAT

Name	Type	Description
null	Trade[]	list of trades filtered by query parameters

A `Trade` object is of the form:

Name	Type	Description
id	string	trade id
taker_order_id	string	hash of taker order (market order) that catalyzed the trade
market	string	market id (condition id)
asset_id	string	asset id (token id) of taker order (market order)
side	string	buy or sell
size	string	size
fee_rate_bps	string	the fees paid for the taker order expressed in basic points
price	string	limit price of taker order
status	string	trade status (see above)
match_time	string	time at which the trade was matched
last_update	string	timestamp of last status update
outcome	string	human readable outcome of the trade
maker_address	string	funder address of the taker of the trade
owner	string	api key of taker of the trade
transaction_hash	string	hash of the transaction where the trade was executed
bucket_index	integer	index of bucket for trade in case trade is executed in multiple transactions

Name	Type	Description
maker_orders	MakerOrder[]	list of maker trades the taker trade was filled against
type	string	side of the trade: TAKER or MAKER

A `MakerOrder` object is of the form:

Name	Type	Description
order_id	string	id of maker order
maker_address	string	maker address of the order
owner	string	api key of the owner of the order
matched_amount	string	size of maker order consumed with this trade
fee_rate_bps	string	the fees paid for the taker order expressed in basic points
price	string	price of maker order
asset_id	string	token/asset id
outcome	string	human readable outcome of the maker order

Markets

GET MARKETS

Get available CLOB markets (paginated).

HTTP REQUEST

`GET {clob-endpoint}/markets?next_cursor={next_cursor}`

```
async function main() {
  const markets = await clobClient.getMarkets("");
  console.log(`markets: `);
  console.log(markets);
}

main();
```

REQUEST PARAMETERS

Name	Required	Type	Description
next_cursor	no	string	cursor to start with, used for traversing paginated response

RESPONSE FORMAT

Name	Type	Description
limit	number	limit of results on a single page

Name	Type	Description
count	number	number of results
next_cursor	string	pagination item to retrieve the next page base64 encoded. 'LTE=' means the end and empty ('') means the beginning
data	Market[]	list of markets

A `Market` object is of the form:

Name	Type	Description
condition_id	string	id of market which is also the CTF condition ID
question_id	string	question id of market which is also the CTF question ID which is used to derive the <code>condition_id</code>
tokens	Tokens[2]	binary token pair for market
rewards	Rewards	rewards related data
minimum_order_size	string	minimum limit order size
minimum_tick_size	string	minimum tick size in units of implied probability (max price resolution)
description	string	market description
category	string	market category
end_date_iso	string	iso string of market end date
game_start_time	string	iso string of game start time which is used to trigger delay
question	string	question
market_slug	string	slug of market
min_incentive_size	string	minimum resting order size for incentive qualification
max_incentive_spread	string	max spread up to which orders are qualified for incentives (in cents)
active	boolean	boolean indicating whether market is active/live
closed	boolean	boolean indicating whether market is closed/open
seconds_delay	integer	seconds of match delay for in-game trading
icon	string	reference to market icon image
fpmm	string	address of associated fixed product market maker on Polygon network

Where the `Token` object is of the form:

Name	Type	Description
token_id	string	erc1155 token id

Name	Type	Description
outcome	string	human readable outcome

Where the `Rewards` object is of the form:

Name	Type	Description
min_size	number	min size of an order to score
max_spread	number	max spread from the midpoint until an order scores
event_start_date	string	string date when the event starts
event_end_date	string	string date when the event end
in_game_multiplier	number	reward multiplier while the game has started
reward_epoch	number	current reward epoch

GET SAMPLING MARKETS

Get available CLOB markets that have rewards enabled.

HTTP REQUEST

`GET {clob-endpoint}/sampling-markets?next_cursor={next_cursor}`

```
async function main() {
  const markets = await clobClient.getSamplingMarkets("");
  console.log(`markets: `);
  console.log(markets);
}

main();
```

RESPONSE FORMAT

Name	Type	Description
limit	number	limit of results on a single page
count	number	number of results
next_cursor	string	pagination item to retrieve the next page base64 encoded. 'LTE=' means the end and empty ('') means the beginning
data	Market[]	list of sampling markets

GET SIMPLIFIED MARKETS

Get available CLOB markets expressed in a reduced schema.

HTTP REQUEST

`GET {clob-endpoint}/simplified-markets?next_cursor={next_cursor}`

```
async function main() {
  const markets = await clobClient.getSimplifiedMarkets("");
  console.log(`markets: `);
}
```

```
    console.log(markets);
  }

  main();
```

RESPONSE FORMAT

Name	Type	Description
limit	number	limit of results on a single page
count	number	number of results
next_cursor	string	pagination item to retrieve the next page base64 encoded. 'LTE=' means the end and empty ('') means the beginning
data	SimplifiedMarket[]	list of markets

A `SimplifiedMarket` object is of the form:

Name	Type	Description
condition_id	string	id of market which is also the CTF condition ID
tokens	Tokens[2]	binary token pair for market
rewards	Rewards	rewards related data
min_incentive_size	string	minimum resting order size for incentive qualification
max_incentive_spread	string	max spread up to which orders are qualified for incentives (in cents)
active	boolean	boolean indicating whether market is active/live
closed	boolean	boolean indicating whether market is closed/open

GET SAMPLING SIMPLIFIED MARKETS

Get available CLOB markets expressed in a reduced schema that have rewards enabled.

HTTP REQUEST

```
GET {clob-endpoint}/sampling-simplified-markets?next_cursor={next_cursor}
```

```
async function main() {
  const markets = await clobClient.getSamplingSimplifiedMarkets("");
  console.log(`markets: `);
  console.log(markets);
}

main();
```

RESPONSE FORMAT

Name	Type	Description
limit	number	limit of results on a single page
count	number	number of results

Name	Type	Description
next_cursor	string	pagination item to retrieve the next page base64 encoded. 'LTE=' means the end and empty ('') means the beginning
data	SimplifiedMarket[]	list of sampling markets

GET MARKET

Get a single CLOB market.

```
async function main() {
  const market = await clobClient.getMarket("condition_id");
  console.log(`market: `);
  console.log(market);
}

main();
```

HTTP REQUEST

GET {clob-endpoint}/markets/{condition_id}

RESPONSE FORMAT

Name	Type	Description
market	Market	market object

Prices and Books

GET BOOK

Get a order book summary for a market.

HTTP REQUEST

GET {clob-endpoint}/book

```
async function main() {
  const resp = await clobClient.getOrderBook(
    "71321045679252212594626385532706912750332728571942532289631379312455583992563"
  );
  console.log(resp);
}

main();
```

REQUEST PARAMETERS

Name	Required	Type	Description
token_id	yes	string	token ID of market to get book for

RESPONSE FORMAT

Name	Type	Description
market	string	condition id
asset_id	string	id of the asset/token
hash	string	hash summary of the orderbook content
timestamp	string	unix timestamp the current book generation in milliseconds (1/1,000 second)
bids	OrderSummary[]	list of bid levels
asks	OrderSummary[]	list of ask levels

A `OrderSummary` object is of the form:

Name	Type	Description
price	string	price
size	string	size

GET BOOKS

Get order book summaries for a set of markets.

HTTP REQUEST

`POST {clob-endpoint}/books`

```
async function main() {
  const YES =
    "71321045679252212594626385532706912750332728571942532289631379312455583992563";
  const NO =
    "52114319501245915516055106046884209969926127482827954674443846427813813222426";

  const orderbooks = await clobClient.getOrderBooks([
    { token_id: YES },
    { token_id: NO },
  ] as BookParams[]);

  console.log(orderbooks);
}
main();
```

REQUEST PARAMETERS

Name	Required	Type	Description
params	yes	BookParams	search params for books

A `BookParams` object is of the form:

Name	Required	Type	Description
token_id	yes	string	token ID of market to get book for

RESPONSE FORMAT

Name	Type	Description
-	Orderbook[]	list orderbooks

A `Orderbook` object is of the form:

Name	Type	Description
market	string	condition id
asset_id	string	id of the asset/token
hash	string	hash summary of the orderbook content
timestamp	string	unix timestamp the current book generation in milliseconds (1/1,000 second)
bids	OrderSummary[]	list of bid levels
asks	OrderSummary[]	list of ask levels

A `OrderSummary` object is of the form:

Name	Type	Description
price	string	price
size	string	size

GET PRICE

Get the price for a market (best bid or best ask).

HTTP REQUEST

GET {clob-endpoint}/price

```

async function main() {
  const YES_TOKEN_ID =
    "71321045679252212594626385532706912750332728571942532289631379312455583992563";
  const NO_TOKEN_ID =
    "52114319501245915516055106046884209969926127482827954674443846427813813222426";

  clobClient
    .getPrice(YES_TOKEN_ID, "buy")
    .then((price: any) => console.log("YES", "BUY", price));
  clobClient
    .getPrice(YES_TOKEN_ID, "sell")
    .then((price: any) => console.log("YES", "SELL", price));
  clobClient
    .getPrice(NO_TOKEN_ID, "buy")
    .then((price: any) => console.log("NO", "BUY", price));
  clobClient
    .getPrice(NO_TOKEN_ID, "sell")
    .then((price: any) => console.log("NO", "SELL", price));
}
main();

```

REQUEST PAYLOAD PARAMETERS

Name	Required	Type	Description
token_id	yes	string	token ID to get price for
side	yes	string	buy or sell

RESPONSE

`{"price": ".513"}`

GET PRICES

Get the prices for a group of markets (best bid or best ask).

HTTP REQUEST

`POST {clob-endpoint}/prices`

```
async function main() {
  const YES =
    "71321045679252212594626385532706912750332728571942532289631379312455583992563";
  const NO =
    "52114319501245915516055106046884209969926127482827954674443846427813813222426";

  const prices = await clobClient.getPrices([
    { token_id: YES, side: Side.BUY },
    { token_id: YES, side: Side.SELL },
    { token_id: NO, side: Side.BUY },
    { token_id: NO, side: Side.SELL },
  ] as BookParams[]);
}
```

REQUEST PAYLOAD PARAMETERS

Name	Required	Type	Description
params	yes	BookParams	search params for books

A `BookParams` object is of the form:

Name	Required	Type	Description
token_id	yes	string	token ID of market to get book for
side	yes	string	BUY or SELL

RESPONSE

`{[asset_id]: {[side]: price}}`

```
{
  "52114319501245915516055106046884209969926127482827954674443846427813813222426": {
    "BUY": "0.49",
    "SELL": "0.5"
  },
  "71321045679252212594626385532706912750332728571942532289631379312455583992563": {
    "BUY": "0.5",
    "SELL": "0.51"
  }
}
```

```
}  
}
```

GET MIDPOINT

Get the midpoint price for a market (halfway between best bid or best ask).

HTTP REQUEST

GET {clob-endpoint}/midpoint

```
async function main() {  
  const resp = client.getMidpoint(  
    (tokenId =  
      "71321045679252212594626385532706912750332728571942532289631379312455583992563")  
    );  
  console.log(resp);  
}  
main();
```

REQUEST PAYLOAD PARAMETERS

Name	Required	Type	Description
token_id	yes	string	token ID market to get price for

RESPONSE

{'mid': '0.55'}

GET MIDPOINTS

Get the midpoint prices for a set of market (halfway between best bid or best ask).

HTTP REQUEST

POST {clob-endpoint}/midpoints

```
async function main() {  
  const YES =  
    "71321045679252212594626385532706912750332728571942532289631379312455583992563";  
  const NO =  
    "52114319501245915516055106046884209969926127482827954674443846427813813222426";  
  
  const midpoints = await clobClient.getMidpoints([  
    { token_id: YES },  
    { token_id: NO },  
  ] as BookParams[]);  
}  
main();
```

REQUEST PAYLOAD PARAMETERS

Name	Required	Type	Description
params	yes	BookParams	search params for books

A BookParams object is of the form:

Name	Required	Type	Description
token_id	yes	string	token ID of market to get book for

RESPONSE

{[asset_id]: mid price}

```
{
  "52114319501245915516055106046884209969926127482827954674443846427813813222426": "0.495",
  "71321045679252212594626385532706912750332728571942532289631379312455583992563": "0.505"
}
```

GET SPREAD

Get the spreads for a market.

HTTP REQUEST

GET {clob-endpoint}/spread

```
async function main() {
  const YES_TOKEN_ID =
    "71321045679252212594626385532706912750332728571942532289631379312455583992563";
  const NO_TOKEN_ID =
    "52114319501245915516055106046884209969926127482827954674443846427813813222426";

  clobClient
    .getSpread(YES_TOKEN_ID)
    .then((spread: any) => console.log("YES", spread));
  clobClient
    .getSpread(NO_TOKEN_ID)
    .then((spread: any) => console.log("NO", spread));
}
main();
```

REQUEST PAYLOAD PARAMETERS

Name	Required	Type	Description
token_id	yes	string	token ID to get the spread for

RESPONSE

{"spread": ".513"}

GET SPREADS

Get the spreads for a set of markets.

HTTP REQUEST

GET {clob-endpoint}/spread

```
async function main() {
  const YES =
    "71321045679252212594626385532706912750332728571942532289631379312455583992563";
  const NO =
    "52114319501245915516055106046884209969926127482827954674443846427813813222426";

  const spreads = await clobClient.getSpreads([
    { token_id: YES },
  ])
```

```
{ token_id: NO },
] as BookParams[]);
}
main();
```

REQUEST PAYLOAD PARAMETERS

Name	Required	Type	Description
params	yes	BookParams	search params for books

A `BookParams` object is of the form:

Name	Required	Type	Description
token_id	yes	string	token ID of market to get book for

RESPONSE

```
{[asset_id]: spread}
```

```
{
  "52114319501245915516055106046884209969926127482827954674443846427813813222426": "0.01",
  "71321045679252212594626385532706912750332728571942532289631379312455583992563": "0.01"
}
```

Websocket API

OVERVIEW

The Polymarket CLOP API provides websocket (wss) channels through which clients can get pushed updates. These endpoints allow clients to maintain almost real-time views of their orders, their trades and markets in general. There are two available channels `user` and `market`.

SUBSCRIPTION

To subscribe send a message including the following authenticating and intent information upon opening the connection.

Field	Type	Description
auth	Auth	see WSS Authentication
markets	string[]	array of markets (condition IDs) to receive events for (for <code>user</code> channel)
assets_ids	string[]	array of asset ids (token IDs) to receive events for (for <code>market</code> channel)
type	string	id of channel to subscribe to (User or Market)

where the `auth` field is of type `Auth` which has the form described in the WSS Authentication section below.

WSS AUTHENTICATION

i Only connections to `user` channel require authentication.

Field	Optional	Description
apiKey	yes	Polygon account's CLOB api key
secret	yes	Polygon account's CLOB api secret
passphrase	yes	Polygon account's CLOB api passphrase

USER CHANNEL

Authenticated channel for updates related to user activities (orders, trades), filtered for authenticated user (by apiKey).

SUBSCRIBE {wss-channel} user

TRADE MESSAGE

```
{
  "asset_id": "52114319501245915516055106046884209969926127482827954674443846427813813222426",
  "event_type": "trade",
  "id": "28c4d2eb-bbea-40e7-a9f0-b2fdb56b2c2e",
  "last_update": "1672290701",
  "maker_orders": [
    {
      "asset_id": "52114319501245915516055106046884209969926127482827954674443846427813813222426",
      "matched_amount": "10",
      "order_id": "0xff354cd7ca7539dfa9c28d90943ab5779a4eac34b9b37a757d7b32bdfb11790b",
      "outcome": "YES",
      "owner": "9180014b-33c8-9240-a14b-bdca11c0a465",
      "price": "0.57"
    }
  ],
  "market": "0xbd31dc8a20211944f6b70f31557f1001557b59905b7738480ca09bd4532f84af",
  "matchtime": "1672290701",
  "outcome": "YES",
  "owner": "9180014b-33c8-9240-a14b-bdca11c0a465",
  "price": "0.57",
  "side": "BUY",
  "size": "10",
  "status": "MATCHED",
  "taker_order_id": "0x06bc63e346ed4ceddce9efd6b3af37c8f8f440c92fe7da6b2d0f9e4ccbc50c42",
  "timestamp": "1672290701",
  "trade_owner": "9180014b-33c8-9240-a14b-bdca11c0a465",
  "type": "TRADE"
}
```

Emitted when:

- when a market order is matched ("MATCHED")
- when a limit order for a user is included in a trade ("MATCHED")
- subsequent status changes for trade ("MINED", "CONFIRMED", "RETRYING", "FAILED")

Structure:

Name	Type	Description
asset_id	string	asset id (token ID) of taker order (market order)
event_type	string	"trade"
id	string	trade id
last_update	string	time of last update to trade

Name	Type	Description
maker_orders	MakerOrder[]	array of maker order match details
market	string	market identifier (condition ID)
matchtime	string	time trade was matched
outcome	string	outcome
owner	string	api key of event owner
price	string	price
side	string	BUY/SELL
size	string	size
status	string	trade status
taker_order_id	string	id of taker order
timestamp	string	time of event
trade_owner	string	api key of trade owner
type	string	"TRADE"

Where a `MakerOrder` object is of the form:

Name	Type	Description
asset_id	string	asset id of the maker order
matched_amount	string	amount of maker order matched in trade
order_id	string	maker order ID
outcome	string	outcome
owner	string	owner of marker order
price	string	price of maker order

`ORDER` MESSAGE

```
{
  "asset_id": "52114319501245915516055106046884209969926127482827954674443846427813813222426",
  "associate_trades": null,
  "event_type": "order",
  "id": "0xff354cd7ca7539dfa9c28d90943ab5779a4eac34b9b37a757d7b32bdfb11790b",
  "market": "0xbd31dc8a20211944f6b70f31557f1001557b59905b7738480ca09bd4532f84af",
  "order_owner": "9180014b-33c8-9240-a14b-bdca11c0a465",
  "original_size": "10",
  "outcome": "YES",
  "owner": "9180014b-33c8-9240-a14b-bdca11c0a465",
  "price": "0.57",
  "side": "SELL",
  "size_matched": "0",
```

```
{
  "timestamp": "1672290687",
  "type": "PLACEMENT"
}
```

Emitted when:

- When an order is placed (PLACEMENT)
- When an order is updated (some of it is matched) (UPDATE)
- When an order is cancelled (CANCELLATION)

Structure:

Name	Type	Description
asset_id	string	asset ID (token ID) of order
associate_trades	string[]	array of ids referencing trades that the order has been included in
event_type	string	"order"
id	string	order id
market	string	condition ID of market
order_owner	string	owner of order
original_size	string	original order size
outcome	string	outcome string
owner	string	owner of order
price	string	price of order
side	string	BUY/SELL
size_matched	string	size of order that has been matched
timestamp	string	time of event
type	string	PLACEMENT/UPDATE/CANCELLATION

MARKET CHANNEL

Public channel for updates related to market updates (level 2 price data).

SUBSCRIBE {wss-channel} market

BOOK MESSAGE

```
{
  "event_type": "book",
  "asset_id": "65818619657568813474341868652308942079804919287380422192892211131408793125422",
  "market": "0xbd31dc8a20211944f6b70f31557f1001557b59905b7738480ca09bd4532f84af",
  "buys": [
    { "price": ".48", "size": "30" },
    { "price": ".49", "size": "20" },
    { "price": ".50", "size": "15" }
  ],
  "sells": [
    { "price": ".52", "size": "25" },

```

```
{ "price": ".53", "size": "60" },
{ "price": ".54", "size": "10" }
],
"timestamp": "123456789000",
"hash": "0x0...."
}
```

Emitted When:

- First subscribed to `market/`
- When there is a trade that affects the book

Structure:

Name	Type	Description
event_type	string	"book"
asset_id	string	asset ID (token ID)
market	string	condition ID of market
timestamp	string	unix timestamp the current book generation in milliseconds (1/1,000 second)
hash	string	hash summary of the orderbook content
buys	OrderSummary[]	list of type {size, price} aggregate book levels for buys
sells	OrderSummary[]	list of type {size, price} aggregate book levels for sells

Where a `OrderSummary` object is of the form:

Name	Type	Description
price	string	size available at that price level
size	string	price of the orderbook level

`PRICE_CHANGE` MESSAGE

```
{
  "asset_id": "71321045679252212594626385532706912750332728571942532289631379312455583992563",
  "changes": [
    {
      "price": "0.4",
      "side": "SELL",
      "size": "3300"
    },
    {
      "price": "0.5",
      "side": "SELL",
      "size": "3400"
    },
    {
      "price": "0.3",
      "side": "SELL",
      "size": "3400"
    }
  ],
  "event_type": "price_change",
  "market": "0x5f65177b394277fd294cd75650044e32ba009a95022d88a0c1d565897d72f8f1",
}
```

```
{
  "timestamp": "1729084877448",
  "hash": "3cd4d61e042c81560c9037ece0c61f3b1a8fbdd"
}
```

Emitted When:

- A new order is placed
- An order is cancelled

Structure:

Name	Type	Description
event_type	string	"price_change"
asset_id	string	asset ID (token ID)
market	string	condition ID of market
price	string	price level affected
size	string	new aggregate size for price level
side	string	buy/sell
timestamp	string	time of event

TICK_SIZE_CHANGE MESSAGE

```
{
  "event_type": "tick_size_change",
  "asset_id": "65818619657568813474341868652308942079804919287380422192892211131408793125422",
  "market": "0xbd31dc8a20211944f6b70f31557f1001557b59905b7738480ca09bd4532f84af",
  "old_tick_size": "0.01",
  "new_tick_size": "0.001",
  "timestamp": "100000000"
}
```

Emitted When:

- The minimum tick size of the market changes. This happens when the book's price reaches the limits: $\text{price} > 0.96$ or $\text{price} < 0.04$

Structure:

Name	Type	Description
event_type	string	"price_change"
asset_id	string	asset ID (token ID)
market	string	condition ID of market
old_tick_size		previous minimum tick size
new_tick_size	string	current minimum tick size
side	string	buy/sell

Name	Type	Description
timestamp	string	time of event

Timeseries Data

The CLOB provides detailed price history data for each traded token.

HTTP REQUEST

GET {clob-endpoint}/prices-history

QUERY PARAMETERS

Name	Type	Description
market	number	the CLOB token id for which to fetch price history
startTs	number	the start time, a unix timestamp in UTC
endTs	number	the end time, a unix timestamp in UTC
interval	string	a string representing a duration ending at the current time
fidelity	number	the resolution of the data, in minutes

The query parameters {startTs/endTs} are mutually exclusive to {interval}. The options for {interval} are:

- {1m}: 1 month
- {1w}: 1 week
- {1d}: 1 day
- {6h}: 6 hours
- {1h}: 1 hour
- {max}: max range

EXAMPLE RESPONSE FORMAT

Name	Type	Description
history	TimeseriesPoint[]	list of timestamp/price pairs

A {TimeseriesPoint} object is of the form:

Name	Type	Description
t	number	utc timestamp
p	number	price

Overview

All market data necessary for market resolution is available on-chain (ie `ancillaryData` in UMA OO request) but Polymarket also provides a hosted service, Gamma, that indexes this data and provides additional market metadata (ie categorization, indexed volume, etc). This service is made available through a REST API. For public users, this resource is read only and can be used to fetch useful information about markets for things like non-profit research projects, alternative trading interfaces, automated trading systems etc.

Endpoint

`https://gamma-api.polymarket.com`

Market Organization

Gamma provides some organizational models. These include **events**, and **markets**. The most fundamental element is always **markets** and the other models simply provide additional organization.

DETAIL

1. Market
 1. Contains data related to a market that is traded on. Maps onto a pair of clob token ids, a market maker address, a question id and a condition id
2. Event
 1. Contains a set of *markets*
 2. Variants:
 1. Event with 1 *market* (i.e. resulting in an SMP)
 2. Event with 2 or more *markets* (i.e. resulting in an GMP)

EXAMPLE

- `[Event]` Where will Barron Trump attend College?
 - `[Market]` Will Barron attend Georgetown?
 - `[Market]` Will Barron attend NYU?
 - `[Market]` Will Barron attend UPenn?
 - `[Market]` Will Barron attend Harvard?
 - `[Market]` Will Barron attend another college?

Markets

Get markets.

Note: Markets can be traded via the CLOB if `enableOrderBook` is `true`.

HTTP REQUEST

`GET {gamma-endpoint}/markets`

QUERY PARAMETERS

Name	Type	Description
limit	number	limit query results
offset	number	pagination offset
order	string	key to sort by
ascending	boolean	sort direction, defaults to true, requires the <code>order</code> parameter
id	number	id of a single market to query, can be used multiple times to fetch multiple markets
slug	string	slug of a single event to query, can be used multiple times to fetch multiple markets
archived	boolean	filter by archived status
active	boolean	filter by active status
closed	boolean	filter by closed status
clob_token_ids	string	filter by clob token id, can be used multiple times
condition_ids	string	filter by condition id, can be used multiple times
liquidity_num_min	decimal	filter by minimum liquidity
liquidity_num_max	decimal	filter by maximum liquidity
volume_num_min	decimal	filter by minimum volume
volume_num_max	decimal	filter by maximum volume
start_date_min	string	filter by minimum start date
start_date_max	string	filter by maximum start date
end_date_min	string	filter by minimum end date
end_date_max	string	filter by maximum end date
tag_id	number	filter by tag id
related_tags	boolean	include events with related tags, requires the <code>tag_id</code> parameter

EXAMPLE QUERIES

```
/markets
/markets?offset=1
/markets?limit=100
/markets?order=slug
/markets?order=slug&ascending=false
/markets?id=1
/markets?id=1&id=2
/markets?slug=market-slug
/markets?slug=market-slug&slug=slug
/markets?archived=true
/markets?closed=true
/markets?active=true
/markets?clob_token_ids=1
/markets?clob_token_ids=1&clob_token_ids=2
```

```
/markets?condition_ids=1
/markets?condition_ids=1&condition_ids=2
/markets?liquidity_num_min=1.0
/markets?liquidity_num_max=2.0
/markets?volume_num_min=1.0
/markets?volume_num_max=2.0
/markets?start_date_min=2022-04-01T00:00:00Z
/markets?start_date_max=2022-04-01T00:00:00Z
/markets?end_date_min=2022-04-01T00:00:00Z
/markets?end_date_max=2022-04-01T00:00:00Z
/markets?tag_id=1&related_tags=true
```

EXAMPLE RESPONSE FORMAT

Name	Type	Description
markets	Market[]	list of markets

Single Market

Get a single event by id.

HTTP REQUEST

GET {gamma-endpoint}/markets/{id}

EXAMPLE QUERY

```
/markets/501011
```

Events

Get events.

HTTP REQUEST

GET {gamma-endpoint}/events

QUERY PARAMETERS

Name	Type	Description
limit	number	limit query results
offset	number	pagination offset
order	string	key to sort by
ascending	boolean	sort direction, defaults to true, requires the <code>order</code> parameter
id	number	id of a single event to query, can be used multiple times to fetch multiple events
slug	string	slug of a single event to query, can be used multiple times to fetch multiple events

Name	Type	Description
archived	boolean	filter by archived status
active	boolean	filter by active status
closed	boolean	filter by closed status
liquidity_min	decimal	filter by minimum liquidity
liquidity_max	decimal	filter by maximum liquidity
volume_min	decimal	filter by minimum volume
volume_max	decimal	filter by maximum volume
start_date_min	string	filter by minimum start date
start_date_max	string	filter by maximum start date
end_date_min	string	filter by minimum end date
end_date_max	string	filter by maximum end date
tag	string	filter by tag labels
tag_id	number	filter by tag id
related_tags	boolean	include events with related tags, requires the <code>tag_id</code> parameter
tag_slug	string	filter by tag slug

i The tags filters are mutually exclusive with priority `tag` > `tag_id` > `tag_slug`.

EXAMPLE QUERIES

```
/events
/events?offset=1
/events?limit=100
/events?order=slug
/events?order=slug&ascending=false
/events?id=1
/events?id=1&id=2
/events?slug=market-slug
/events?slug=market-slug&slug=slug
/events?archived=true
/events?closed=true
/events?active=true
/events?clob_token_ids=tokenid
/events?condition_ids=condition
/events?liquidity_min=1.0
/events?liquidity_max=2.0
/events?volume_min=1.0
/events?volume_max=2.0
/events?start_date_min=2022-04-01T00:00:00Z
/events?start_date_max=2022-04-01T00:00:00Z
/events?end_date_min=2022-04-01T00:00:00Z
/events?end_date_max=2022-04-01T00:00:00Z
/events?tag=music
```

```
/events?tag_id=1
/events?tag_id=1&related_tags=true
```

EXAMPLE RESPONSE FORMAT

Name	Type	Description
events	Event[]	list of events

Single Event

Get a single event by id.

HTTP REQUEST

```
GET {gamma-endpoint}/events/{id}
```

EXAMPLE QUERY

```
/events/901289
```

Subgraph

Overview

Polymarket has written and open sourced a subgraph that provides, via a GraphQL query interface, useful aggregate calculations and event indexing for things like trade, volume, user position, market and liquidity data. The subgraph updates in real time as blocks are mined, and is reorg-aware. The subgraph underlies the primary Polymarket.com interface, providing positional data, activity history and more. The subgraph can be hosted by anyone but is also hosted and made publicly available by a 3rd party provider, Goldsky.

Source

The Polymarket subgraph is entirely open source and can be found on the Polymarket Github.

```
Polymarket/polymarket-subgraph
```

Note: The available models/schemas can be found in the `schema.graphql` file.

Hosted Version

The subgraphs are hosted on goldsky, each with an accompanying GraphQL playground:

- Orders subgraph: https://api.goldskey.com/api/public/project_cl6mb8i9h0003e201j6li0diw/subgraphs/polymarket-orderbook-resync/prod/gn
- Positions subgraph: https://api.goldskey.com/api/public/project_cl6mb8i9h0003e201j6li0diw/subgraphs/positions-subgraph/0.0.7/gn
- Activity subgraph: https://api.goldskey.com/api/public/project_cl6mb8i9h0003e201j6li0diw/subgraphs/activity-subgraph/0.0.4/gn
- Open Interest subgraph: https://api.goldskey.com/api/public/project_cl6mb8i9h0003e201j6li0diw/subgraphs/oi-subgraph/0.0.6/gn
- PNL subgraph: https://api.goldskey.com/api/public/project_cl6mb8i9h0003e201j6li0diw/subgraphs/pnl-subgraph/0.0.14/gn

Resolution

All market resolution is completely decentralized. A majority of markets on Polymarket are resolved via UMA's optimistic oracle, the rest (some price markets) are resolved via Pyth.

UMA

OVERVIEW

Polymarket leverages UMA's Optimistic Oracle (OO) to resolve arbitrary questions, permissionlessly. From UMA's docs:

"UMA's Optimistic Oracle allows contracts to quickly request and receive data information. The Optimistic Oracle acts as a generalized escalation game between contracts that initiate a price request and UMA's dispute resolution system known as the Data Verification Mechanism (DVM). Prices proposed by the Optimistic Oracle will not be sent to the DVM unless it is disputed. If a dispute is raised, a request is sent to the DVM. All contracts built on UMA use the DVM as a backstop to resolve disputes. Disputes sent to the DVM will be resolved within a few days - after UMA tokenholders vote on what the correct outcome should have been."

To allow CTF markets to be resolved via the OO, Polymarket developed a custom adapter contract called `UmaCtfAdapter` that provides a way for the two contract systems to interface.

CLARIFICATIONS

Recent versions (v2+) of the `UmaCtfAdapter` also include a bulletin board feature that allows market creators to issue "clarifications". Questions that allow updates will include the sentence in their ancillary data:

"Updates made by the question creator via the bulletin board at 0x6A9D222616C90FcA5754cd1333cFD9b7fb6a4F74 as described by <https://polygonscan.com/tx/0xa14f01b115c4913624fc3f508f960f4dea252758e73c28f5f07f8e19d7bca066> should be considered"

Where the transaction referenced includes a message outlining what should and shouldn't be considered. In summary, only clarifications that do not impact the question's intent should be considered.

RESOLUTION PROCESS

ACTIONS:

- **Initialize** - Binary CTF markets are initialized via the `UmaCtfAdapter`'s `initialize()` function. This stores the question parameters on the contract, prepares the question on the CTF and requests a price for a question from the OO. It returns a CTF `questionID` that is also used as reference on the `UmaCtfAdapter`. The caller provides:
 1. `ancillaryData` - data used to resolve a question (ie the question + description)

2. `rewardToken` - ERC20 token address used for payment of rewards and fees
 3. `reward` - Reward amount offered to a successful proposer. The caller must have set allowance so that the contract can pull this reward in.
 4. `proposalBond` - Bond required to be posted by OO proposers/disputers. If 0, the default OO bond is used.
 5. `liveness` - UMA liveness period in seconds. If 0, the default liveness period is used.
- **Propose Price** - Anyone can then propose a price to the question on the OO. To do this they must post the `proposalBond`. The liveness period begins after a price is proposed.
 - **Dispute** - Anyone that disagrees with the proposed price has the opportunity to dispute the price by posting a counter bond via the OO, this price proposal will now be escalated to the DVM for a voter-wide vote.

POSSIBLE FLOWS:

When the first proposed price is disputed for a `questionId` on the adapter, a callback is made that resets the `questionId`, making a new request to the OO (the reward is returned before the callback is made and posted as the reward for this new proposal). This means a second proposal is required for a `questionId` to resolve, and correspondingly a second dispute is required for it to go to the DVM. The thinking behind here is it doubles the cost of a potential grieving vector (two disputes are required versus just one) and also allows fat-fingered (incorrect) first price proposals to not delay resolution. As such we have the following possible resolution flows:

- Initialize (CTFAdaptor) -> Propose (OO) -> Resolve (CTFAdaptor)
- Initialize (CTFAdaptor) -> Propose (OO) -> Challenge (OO) -> Propose (OO) -> Resolve (CTFAdaptor)
- Initialize (CTFAdaptor) -> Propose (OO) -> Challenge (OO) -> Propose (OO) -> Challenge (CtfAdapter) -> Resolve (CTFAdaptor)

DEPLOYED ADDRESSES

V3.0.0

Network	Address
Polygon Mainnet	0x71392E133063CC0D16F40E1F9B60227404Bc03f7
Mumbai	0x71392E133063CC0D16F40E1F9B60227404Bc03f7

V2.0.0

Network	Address
Polygon Mainnet	0x6A9D222616C90FcA5754cd1333cFD9b7fb6a4F74
Mumbai	0x6A9D222616C90FcA5754cd1333cFD9b7fb6a4F74

V1.0.0

Network	Address
Polygon Mainnet	0xCB1822859cEF82Cd2Eb4E6276C7916e692995130
Mumbai	0xCB1822859cEF82Cd2Eb4E6276C7916e692995130

ADDITIONAL RESOURCES

- Audit
- Source Code
- UMA Documentation
- UMA Oracle Portal

Rewards

Polymarket provides incentives aimed at catalyzing the supply and demand side of the marketplace. Specifically there is a public liquidity rewards program as well as one-off public pnl/volume competitions.

Liquidity Rewards Program

OVERVIEW

By posting resting limit orders, liquidity providers (makers) are automatically eligible for Polymarket's incentive program. The overall goal of this program is to **catalyze a healthy, liquid marketplace. We can further define this as creating incentives that:**

- Catalyze liquidity across all markets
- Encourage liquidity throughout a market's entire lifecycle
- Motivate passive, balanced quoting tight to a market's mid point
- Encourages trading activity
- Discourages blatantly exploitative behaviors

This program is heavily inspired by dYdX's liquidity provider rewards which you can read more about here. In fact, the incentive methodology is essentially a copy of dYdX's successful methodology but with some adjustments including specific adaptations for binary contract markets with distinct books, no staking mechanic a slightly modified order utility-relative to depth function and reward amounts isolated per market. Rewards are distributed directly to maker's addresses daily at midnight UTC.

METHODOLOGY

Polymarket liquidity providers will be rewarded based on a formula that rewards participation in markets (complementary consideration), boosts two-sided depth (single-sided orders still score), and spread (vs. mid-market, adjusted for the size cutoff*). Each market will configure a max spread and min size cutoff within which orders are considered. The amount of rewards earned is determined by the relative share of each participant's Q_{epoch} in each market multiplied by the rewards available for that market.

Variable	Description
S	order position scoring function
v	max spread from midpoint (in cents)
s	spread from size-cutoff-adjusted midpoint
b	in-game multiplier
m	market
m'	market complement (ie NO if m = YES)
n	trader index
u	sample index
c	scaling factor (currently 3.0 on all markets)

Variable	Description
Q_{one}	point total for book one for a sample
Q_{two}	point total for book two for a sample
Spread_{m_n}	distance from midpoint (bps or relative) for order n in market m
BidSize	share-denominated quantity of bid
AskSize	share-denominated quantity of ask

Equation 1:

$$S(v, s) = \left(\frac{v - s}{v} \right)^2 \cdot b$$

Equation 2:

$$Q_{\text{one}} = S(v, \text{Spread}_{m_1}) \cdot \text{BidSize}_{m_1} + S(v, \text{Spread}_{m_2}) \cdot \text{BidSize}_{m_2} + \dots \\ + S(v, \text{Spread}_{m'_1}) \cdot \text{AskSize}_{m'_1} + S(v, \text{Spread}_{m'_2}) \cdot \text{AskSize}_{m'_2}$$

Equation 3:

$$Q_{\text{two}} = S(v, \text{Spread}_{m_1}) \cdot \text{AskSize}_{m_1} + S(v, \text{Spread}_{m_2}) \cdot \text{AskSize}_{m_2} + \dots \\ + S(v, \text{Spread}_{m'_1}) \cdot \text{BidSize}_{m'_1} + S(v, \text{Spread}_{m'_2}) \cdot \text{BidSize}_{m'_2}$$

Equation 4:

Equation 4a:

If midpoint is in range [0.10,0.90] allow single sided liq to score:

$$Q_{\min} = \max(\min(Q_{\text{one}}, Q_{\text{two}}), \max(Q_{\text{one}}/c, Q_{\text{two}}/c))$$

Equation 4b:

If midpoint is in either range [0,0.10) or (.90,1.0] require liq to be double sided to score:

$$Q_{\min} = \min(Q_{\text{one}}, Q_{\text{two}})$$

Equation 5:

$$Q_{\text{normal}} = \frac{Q_{\min}}{\sum_{n=1}^N (Q_{\min})_n}$$

Equation 6:

$$Q_{\text{epoch}} = \sum_{u=1}^{10,080} (Q_{\text{normal}})_u$$

Equation 7:

$$Q_{\text{final}} = \frac{Q_{\text{epoch}}}{\sum_{n=1}^N (Q_{\text{epoch}})_n}$$

Equation	Description/Explanation
1	Quadratic scoring rule for an order based on position between the adjusted mid point and the maximum qualifying spread
2	<p>Calculates first market side score. Assume a trader has the following open orders:</p> <ul style="list-style-type: none"> - 100q bid on m @0.49 (adjusted midpoint is 0.50 then spread of this order is .01 or 1c) - 200q bid on m @0.48 - 100q ask on m' @0.51 <p>and assume an adjusted market midpoint of 0.50 and maxSpread config of 3c for both m and m'. Then the trader's score is:</p> $Q_{\text{one}} = ((3-1)/3)^2 _ 100 + ((3-2)/3)^2 _ 200 + ((3-1)/3)^2 _ 100$ <p>Q_{one} is calculated every minute using random sampling</p>
3	<p>Calculates second market side score. Assume a trader has the following open orders:</p> <ul style="list-style-type: none"> - 100q bid on m @0.485 - 100q bid on m' @0.48 - 200q ask on m' @0.505 <p>and assume an adjusted market midpoint of 0.50 and maxSpread config of 3c for both m and m'. Then the trader's score is:</p> $Q_{\text{two}} = ((3-1.5)/3)^2 _ 100 + ((3-2)/3)^2 _ 100 + ((3-.5)/3)^2 _ 200$ <p>Q_{two} is calculated every minute using random sampling</p>
4	<p>Boosts 2-sided liquidity by taking the minimum of Q_{one} and Q_{two} and rewards 1-side liquidity at a reduced rate (divided by c)</p> <p>Calculated every minute</p>
5	Q_{normal} is the Q_{min} of a market maker divided by the sum of all the Q_{min} of other market makers in a given sample
6	Q_{epoch} is the sum of all Q_{normal} for a trader in a given epoch
7	Q_{final} normalizes Q_{epoch} by dividing it by the sum of all other market maker's Q_{epoch} in a given epoch this value is multiplied by the rewards available for the market to get a trader's reward

* The adjusted midpoint is a size-cutoff adjusted interpretation of a traditional midpoint $((\text{best_bid} + \text{best_ask})/2)$ which is calculated using the equation $(\text{best_bid_min_shares_adjusted} + \text{best_ask_min_shares_adjusted})/2$ where the $\text{best_bid_min_shares_adjusted}$ is the price level at which the cumulative size of open bids, at or better than that level is greater than or equal to the min shares requirement and the $\text{best_ask_min_shares_adjusted}$ is the price level at which the cumulative size of open asks, at or better than that level is greater than or equal to the min shares requirement.

MARKET CONFIGURATIONS

Both `min_incentive_size` and `max_incentive_spread` can be fetched alongside full market objects via both the CLOB API and Markets API. Reward allocations for an epoch can be fetched via the Markets API.

Leaderboard Competitions

coming soon

Conditional Tokens Framework

Overview

All outcomes on Polymarket are tokenized on the Polygon network. Specifically, Polymarket outcome shares are binary outcomes (ie "YES" and "NO") represented using Gnosis' Conditional Token Framework (CTF). They are distinct ERC1155 tokens related to a parent condition and backed by some collateral. More technically, the binary outcome tokens are referred to as "positionIds" in Gnosis's documentation. "PositionIds" are derived from a collateral token and distinct "collectionIds". "CollectionIds" are derived from a "parentCollectionId", (always bytes32(0) in our case) a "conditionId", and a unique "indexSet". The "indexSet" is a 256 bit array denoting which outcome slots are in an outcome collection; it must be a nonempty proper subset of a condition's outcome slots. In the binary case, which we are interested in, there are two "indexSets", one for the first outcome and one for the second. The first outcome's "indexSet" is 0b01 == 1 and the second's is 0b10 == 2. The parent "conditionId" (shared by both "collectionIds" and therefore "positionIds") is derived from a "questionId" (a hash of the UMA ancillary data), an "oracle" address (the UMA adapter V2) and an "outcomeSlotCount" (always 2 in the binary case). The steps for calculating the ERC1155 token ids (positionIds) is as follows:

1. Get the `conditionId`
 1. **Function:**
 1. `getConditionId(oracle, questionId, outcomeSlotCount)`
 2. **Inputs:**
 1. `oracle`: address - UMA adapter V2
 2. `questionId`: bytes32 - hash of the UMA ancillary data
 3. `outcomeSlotCount`: uint - 2 for binary markets
2. Get the two `collectionIds`
 1. **Function:**
 1. `getCollectionId(parentCollectionId, conditionId, indexSet)`
 2. **Inputs:**
 1. `parentCollectionId`: bytes32 - bytes32(0)
 2. `conditionId`: bytes32 - the conditionId derived from (1)
 3. `indexSet`: uint - 1 (0b01) for the first and 2 (0b10) for the second.
3. Get the two `positionIds`
 1. **Function:**
 1. `getPositionId(collateralToken, collectionId)`
 2. **Inputs:**
 1. `collateralToken`: IERC20 - address of ERC20 token collateral (USDC)
 2. `collectionId`: bytes32 - the two collectionIds derived from (3)

Leveraging the relations above, specifically "conditionIds" → "positionIds" the Gnosis CTF contract allows for "splitting" and "merging" full outcome sets. We explore these actions and provide code examples below.

Split

At any time, after a condition has been prepared on the CTF contract (via `prepareCondition`), it is possible to "split" collateral into a full (position) set. In other words, one unit USDC can be split into 1 YES unit and 1 NO unit. If splitting from the collateral, the CTF contract will attempt to transfer `amount` collateral from the message sender to itself. If successful, `amount` stake will be minted in the split target positions. If any of the transfers, mints, or burns fail, the transaction will revert. The transaction will also revert if the given partition is trivial, invalid, or refers to more slots than the condition is prepared with. This operation happens via the `splitPosition()` function on the CTF contract with the following parameters:

- `collateralToken`: IERC20 - The address of the positions' backing collateral token.

- `parentCollectionId`: bytes32 - The ID of the outcome collections common to the position being split and the split target positions. Null in Polymarket case.
- `conditionId`: bytes32 - The ID of the condition to split on.
- `partition`: uint[] - An array of disjoint index sets representing a nontrivial partition of the outcome slots of the given condition. E.g. A|B and C but not A|B and B|C (is not disjoint). Each element's a number which, together with the condition, represents the outcome collection. E.g. 0b110 is A|B, 0b010 is B, etc. In the Polymarket case 1,2.
- `amount` - The amount of collateral or stake to split. Also the number of full sets to receive.

Merge

In addition to splitting collateral for a full set, the inverse can also happen; a full set can be "merged" for collateral. This operation can again happen at any time after a condition has been prepared on the CTF contract. One unit of each position in a full set is burned in return for 1 collateral unit. This operation happens via the `mergePosition()` function on the CTF contract with the following parameters:

- `collateralToken`: IERC20 - The address of the positions' backing collateral token.
- `parentCollectionId`: bytes32 - The ID of the outcome collections common to the position being merged and the merge target positions. Null in Polymarket case.
- `conditionId`: bytes32 - The ID of the condition to merge on.
- `partition`: uint[] - An array of disjoint index sets representing a nontrivial partition of the outcome slots of the given condition. E.g. A|B and C but not A|B and B|C (is not disjoint). Each element's a number which, together with the condition, represents the outcome collection. E.g. 0b110 is A|B, 0b010 is B, etc. In the Polymarket case 1,2.
- `amount` - The number of full sets to merge. Also the amount of collateral to receive.

Redeem

Once a condition has had its payouts reported (ie by the UmaCTFAdapter calling `reportPayouts` on the CTF contract), users with shares in the winning outcome can redeem them for the underlying collateral. Specifically, users can call the `redeemPositions` function on the CTF contract which will burn all valuable conditional tokens in return for collateral according to the reported payout vector. This function has the following parameters:

- `collateralToken`: IERC20 - The address of the positions' backing collateral token.
- `parentCollectionId`: bytes32 - The ID of the outcome collections common to the position being redeemed. Null in Polymarket case.
- `conditionId`: bytes32 - The ID of the condition to redeem.
- `indexSets`: uint[] - An array of disjoint index sets representing a nontrivial partition of the outcome slots of the given condition. E.g. A|B and C but not A|B and B|C (is not disjoint). Each element's a number which, together with the condition, represents the outcome collection. E.g. 0b110 is A|B, 0b010 is B, etc. In the Polymarket case 1,2.

Deployment

The CTF contract is deployed (and verified) at the following addresses:

Network	Deployed Address
Polygon Mainnet	<code>0x4D97DCd97eC945f40cF65F87097ACe5EA0476045</code>
Mumbai	<code>0x7D8610E9567d2a6C9FBf66a5A13E9Ba8bb120d43</code>

Resources

- Source Code
- Audits
- Gnosis CTF Technical Documentation
- Gist For `positionId` Calculation

FPMMs

Overview

Underlying all markets, and providing for an alternative trading mechanism to the central limit order book (CLOB) are fixed product market makers (FPMMs). These FPMMs are deployed upon market creation using a factory contract. As such, each market has a unique FPMM deployment/address. The addresses and associations of these deployments are indexed in the subgraph and made available via the Gamma API. The contracts themselves are also self-descriptive. As the name suggests, FPMMs implement a fixed (ie "constant") product scoring rule wherein the product of the tokens in the pool (ie the liquidity) is held constant. By maintaining this invariant, the price of a trade is calculated. Importantly, the tokens in the pool are outcome shares. Anyone can provide liquidity to the contract, and in return get ERC20 tokens representing their share of the liquidity pool. When users trade against the pool they pay a configured fee, in collateral, which is reserved proportionally for liquidity providers. When a trader buys shares from a pool, their collateral is split into the underlying conditional tokens, the token they are buying is kept and the rest are sold to the pool for more of the token they are buying. When selling, the user is actually trading the token for equal amounts of the other tokens in the pool (in the binary case just one other outcome token) and then the full sets are merged for collateral. The source implementation of the fixed product was developed by Gnosis.

Add Liquidity

Liquidity providers can provide liquidity to the pool in the form of collateral via `addFunding()`. The function has the following parameters:

- `addedFunds`: uint - The amount of collateral to add to the pool in return for tokens representing their share of the pool and fees.
- `distributionHint`: uint[] - Array indicating the ratios of outcome tokens for initial liquidity. This can only be provided by the first liquidity provider in the pool. [1,1] in the 50/50 starting case.

Note: this is not front-run protected so should be used with a contract that enforces some protection against front running.

Remove Liquidity

At any time, liquidity providers can remove liquidity from a pool they are providing to via `removeFunding()` with the parameter:

- `sharesToBurn`: uint - The number of liquidity shares to burn. The underlying conditional tokens are returned to the liquidity provider along with any earned fees (in collateral).

Buy

Traders can buy outcome tokens from the pool via the `buy()` function with the following parameters:

- `investmentAmount`: uint - The amount of collateral to spend.
- `outcomeIndex`: uint - The index of the outcome to purchase.
- `minOutcomeTokensToBuy`: unit - The minimum amount of tokens you expect to receive.

Sell

Traders can sell outcome tokens to the pool via the `sell()` function with the following parameters:

- `returnAmount`: uint - The amount of collateral to get back.
- `outcomeIndex`: uint - The index of the outcome to sell.
- `maxOutcomeTokensToSell`: uint - The maximum number of outcome tokens to sell to get the `returnAmount` of collateral.

Deployments

FPMM Factory Deployments

Network	Address
Polygon	<code>0x8b9805a2f595b6705e74f7310829f2d299d21522</code>
Mumbai	<code>0x5a80a64c8ae0551b09c3df377b60953872c5aa7f</code>

Resources

- [Source Code](#)
- [Audit](#)

Proxy Wallets

Overview

When a user first uses Polymarket.com to trade they are prompted to create a wallet. When they do this, a 1 of 1 multisig is deployed to Polygon which is controlled/owned by the accessing EOA (either MetaMask wallet or MagicLink wallet). This proxy wallet is where all the user's positions (ERC1155) and USDC (ERC20) are held. Using proxy wallets allows Polymarket to provide an improved UX where multi-step transactions can be executed atomically and transactions can be relayed by relayers on the gas station network. If you are a developer looking to programmatically access positions you accumulated via the Polymarket.com

interface, you can either continue using the smart contract wallet by executing transactions through it from the owner account, or you can transfer these assets to a new address using the owner account.

Deployments

Each user has their own proxy wallet (and thus proxy wallet address) but the factories are available at the following deployed addresses on the **Polygon network**.

Address	Details
<div>0xaacfee03eb1561c4e67d661e40682bd20e3541b</div>	Gnosis safe factory - Gnosis safes are used for all MetaMask users
<div>0xaB45c5A4B0c941a2F231C04C3f49182e1A254052</div>	Polymarket proxy factory - Polymarket custom proxy contracts are used for all MagicLink users

Negative Risk

Overview

Certain events which meet the criteria of being "winner-take-all" may be deployed as "negative risk" events/markets. The Gamma API includes a boolean field on events, `negRisk`, which indicates whether the event is negative risk. Negative risk allows for increased capital efficiency by relating all markets within events via a convert action. More explicitly a NO share in any market can be converted into 1 YES share in all other markets. Converts can be exercised via the Negative Adapter. You can read more about negative risk here.

Augmented Negative Risk

There is a known issue with the negative risk architecture which is that the outcome universe must be complete before conversions are made or otherwise conversion will "cost" something. In most cases the outcome universe can be made complete by deploying all the named outcomes and then an "other" option, but in some cases this is undesirable as new outcomes can come out of nowhere and you'd rather them be directly named versus grouped together in an "other". To fix this, some markets use a system of "augmented negative risk", where named outcomes, a collection of unnamed outcomes and an other is deployed. When a new outcome needs to be added, an unnamed outcome can be clarified to be the new outcome via the bulletin board. This means the "other" in the case of augmented negative risk can effectively change definitions (outcomes can be taken out of it). As such, trading should only happen on the named outcomes, and the other outcomes should be ignored until they are named or until resolution occurs. The Polymarket UI will not show unnamed outcomes. If a market becomes resolvable and the correct outcome is not named (originally or via placeholder clarification) it should resolve to the "other" outcome. An event can be considered "augmented negative risk" when `enableNegRisk` is true AND `negRiskAugmented` is true. The naming conventions are as follows.

Original Outcomes

- Outcome A
- Outcome B
- ...

Placeholder Outcomes

- Person A -> can be clarified to a named outcome
- Person B > can be clarified to a named outcome
- ...

Explicit Other

- Other -> not meant to be traded as the definition of this changes as placeholder outcomes are clarified to named outcomes

Bug Bounty

Polymarket has an active bug bounty program via Immunefi. All public contracts as well as critical web apps are in scope. Please direct any/all reports to the Immunefi program.

Frequently Asked Questions

CLOB Client Questions

HOW DO I INITIALIZE THE CLOB CLIENT?

Users have 3 ways of initializing the CLOB client:

IF TRADING DIRECTLY FROM AN EOA ADDRESS:

Where:

- host: the Polymarket CLOB endpoint <https://clob.polymarket.com>
- key: The private key for the EOA that holds funds
- chain_id: The chain ID for blockchain. 137 in most cases

IF TRADING FROM AN ACCOUNT WITH EMAIL/MAGIC LOGIN:

Where:

- host: the Polymarket CLOB endpoint <https://clob.polymarket.com>
- key: The private key exported from Magic that is associated with the account
- chain_id: The chain ID for blockchain. 137 in most cases
- funder: The Polymarket address associated with the email login

IF TRADING FROM AN ACCOUNT WITH BROWSER WALLET (METAMASK, COINBASE WALLET)

Where:

- host: the Polymarket CLOB endpoint <https://clob.polymarket.com>
- key: The private key exported from the Browser Wallet that is associated with the account
- chain_id: The chain ID for blockchain. 137 in most cases

- funder: The Polymarket address associated with the browser wallet login

See CLOB documentation and these links for more information

Onchain questions

HOW DO I INTERPRET THE ORDERFILLED ONCHAIN EVENT?

Given an OrderFilled event:

- orderHash: a unique hash for the Order being filled
- maker: the user generating the order and the source of funds for the order
- taker: the user filling the order OR the Exchange contract if the order fills multiple limit orders
- makerAssetId: id of the asset that is given out. If 0, indicates that the Order is a BUY, giving USDC in exchange for Outcome tokens. Else, indicates that the Order is a SELL, giving Outcome tokens in exchange for USDC.
- takerAssetId: id of the asset that is received. If 0, indicates that the Order is a SELL, receiving USDC in exchange for Outcome tokens. Else, indicates that the Order is a BUY, receiving Outcome tokens in exchange for USDC.
- makerAmountFilled: the amount of the asset that is given out.
- takerAmountFilled: the amount of the asset that is received.
- fee: the fees paid by the order maker