# MollyBet API

# Contents:

**1 Introduction** **4**
**2 Overview** **5**
2.1 General Workflow 5
2.2 Price and Stake 6
2.3 Order vs Bet 6
**3 API Requests Structure** **7**
3.1 Synchronous Requests Format 7
3.2 Synchronous Responses Format 8
3.3 Creating an Asynchronous Stream 9
3.4 Asynchronous Server Message Format 9
3.5 Asynchronous Client Message Format 10
3.6 Examples 10
**4 Available Requests** **16**
4.1 Sessions Requests 17
4.2 Betslips Requests 19
4.3 Orders Requests 24
4.4 Positions Requests 36
4.5 Customers Requests 37
4.6 Groups Requests 43
4.7 Transfers Requests 45
4.8 Miscellaneous Requests 46
**5 Asynchronous Client Message Types** **48**
5.1 Messages 48
**6 Asynchronous Server Message Types** **50**
6.1 Connection Messages 51
6.2 Event Messages 52
6.3 Betslips Messages 57
6.4 Orders Messages 59
6.5 Other Messages 60
**7 Appendix** **61**
7.1 List of Sports 61
7.2 List of Bet Types 61
7.3 List of Bet Statuses 66
7.4 List of Order Statuses 67
7.5 List of Order Log Categories 67
7.6 Glossary 68

8 Frequently Asked Questions 69
9 Legal 71

# 1 Introduction

MollyBet is the most advanced liquidity pooling execution platform designed exclusively for professional high staking traders, to give them the edge in an increasingly sophisticated market.

The MollyBet website is built on top of *MollyBet API*. MollyBet API provides a means of programmatically accessing a superset of the MollyBet website's functionality.

# 2 Overview

Communication with MollyBet API servers happens over two distinct channels:

1. **Synchronous** request-response HTTPS calls, for one-off operations such as opening betslips, placing orders, and checking the status of bookie accounts.

   These are performed by making a request to a URL and providing either query parameters, or a request body in JSON format depending on the HTTP method used.

   Please refer to the Synchronous Requests Format section for more details about the response structure, available routes, and error handling.

2. **Asynchronous** (or "push") notifications over a persistent WebSocket stream, for immediate state updates, including but not limited to: live betslips, open orders, events, or prices.

   These are received as a sequence of JSON-encoded messages which, when processed in order, update the state of the client to reflect the latest state on the MollyBet servers.

   The Creating an Asynchronous Stream section describes how to connect to the asynchronous stream, the format of the messages, and how to tell MollyBet what data you'd like to receive.

## 2.1 General Workflow

The first step in connecting to MollyBet API is to ensure that your MollyBet account is authorised to access the API. If you have not signed a contract for MollyBet API or have not yet been granted access, you will be unable to login - if that is the case please do not hesitate to get in touch with our support staff.

Once you have been granted access, you may log in by simply:

1. Making an HTTPS call with your MollyBet credentials to create a new session (see Available Requests), and
2. Attaching an asynchronous stream to the session (see Creating an Asynchronous Stream).

As soon as the connections are established, you will start receiving data from the asynchronous stream, such as the current currency conversion rates, your MollyBet balance, and the list of events MollyBet currently knows about.

This is roughly equivalent to loading the MollyBet website Trade page, and only looking at the side-bar (which shows your balance and a list of known competitions). In fact, anyone who has previously used the website will already be familiar with the typical usage scenario of the API:

1. **Register to one or more events.**

   To receive live scores and price updates from your available bookies. This is equivalent to clicking on a competition in the side-bar to show the expanded competition view with detailed prices. Of course, MollyBet API sends raw prices, while the website shows a summarised form.

2. **Inspect the async updates.**

   To find events and prices that look interesting.

3. **Open betslips**.

   To get the exact prices from the bookie accounts you have available.

4. **Place orders**.

   When the prices on the betslip are up to your expectations, you can ask MollyBet to start placing bets, by providing a minimum price and maximum stake.

5. **Receive further bet updates**.

   MollyBet will continue to report the status of placed bets as long as the bookie continues to update them. This is the same information displayed in the "Betbar" on the website.

6. **See step 2**.

## 2.2 Price and Stake

All prices are expressed in decimal odds, in terms of the multiple of your stake that you will win. So a price of 1.95 represents a bet at slightly less than evens and means that you will get back 1.95 times your stake should the bet win (your initial stake plus 0.95 times your stake in winnings). Note that still convert to a price of the form: "1 + your_winnings / your_risk", even when placing a lay bet on the exchange.

Stakes are represented as list of two items, the first element being a string containing the currency code and the second element a number with the amount in that currency.

The currency code is ISO 4217 with the following exceptions:

1. Instead of `"IDR"` we employ `"kIDR"`: 1 kIDR = 1000 Indonesian Rupiahs.

For example, the JSON value `["EUR", 100]` represents the amount of one hundred euros.

## 2.3 Order vs Bet

In the context of MollyBet API, an *Order* represents a request for placing bets on behalf of the client. It includes, among other options, a desired stake and a minimum price. When an *Order* is created, MollyBet API will attempt to place one or more bets with a bookmaker.

Bets that are placed with bookmakers are not placed atomically. Mollybet API tries its best to get bet placement to be confirmed with a bookie, and the bet `status` field (see bet status) is continously updated to reflect what is the degree of certainty that bet has been placed with the bookmaker, or even whether the bet has been settled.

The Mollybet API *Order* also has its own independent `status` field, as well as a `closed` field. The order `closed` field is initially *false* as long as Mollybet API is still trying to fulfil the order, i.e. not all of the desired stake has been confirmed by placed bets. The `closed` field becomes *true* when (a) all the stake that the customer requested has been provided by confirmed bets belonging to that order, (b) the order has expired, or (c) the client requested cancellation. The `status` field (see order status) attempts to summarise the stage of completion and settlement of an order, according to the bets placed and the `closed` field.

Please note that **once a bet has been created, it cannot be cancelled**. The status of a bet evolves indepently of the *Order* that created it. When an *Order* is closed, Mollybet API will stop trying to place more bets, but it will not cancel existing bets already placed.

The only exception to the above rule is regarding putting up orders on exchanges. When Mollybet API finds that there is no liquidity available in any bookmaker at the desired price, it may put up an order in an exchange, to be possibly matched in the future. These exchange order bets, as long as they are not yet matched, can be cancelled by cancelling the *Order* that created them.

# 3 API Requests Structure

**3.1 Synchronous Requests Format** **7**
3.1.1 HTTP persistent connections 8
**3.2 Synchronous Responses Format** **8**
3.2.1 HTTP Status Codes 8
3.2.2 Envelope 8
3.2.3 Rate Limits 9
**3.3 Creating an Asynchronous Stream** **9**
**3.4 Asynchronous Server Message Format** **9**
**3.5 Asynchronous Client Message Format** **10**
**3.6 Examples** **10**
3.6.1 Login example 10
3.6.2 Synchronous request example 11
3.6.3 Asynchronous connection 11
3.6.4 Receiving event offers 12
3.6.5 Working with a betslip 13

We have made every effort to design an API that abides by the HTTP standard, and use widely-supported payload formats such as JSON. In addition, we adhere to REST design principles where practical.

The greatest advantage of following widely adopted standards is the availability of comprehensive client libraries in a wide variety of programming languages. In doing so, we hope to minimise the initial cost of integrating MollyBet API with your own software.

## 3.1 Synchronous Requests Format

In order to send commands to MollyBet API you can perform HTTPS requests to:

```
https://api.mollybet.com/
```

Different requests are exposed as different paths in the API URL domain. Some API requests may accept different HTTP methods (`GET`, `POST`, `DELETE`...) in order to perform different operations.

Available Requests explains the accepted methods and their semantics. `GET` requests only return data from the server and are guaranteed to never have side effects. Other methods may change the system state (e.g. placing or canceling an order).

Several API requests accept optional or mandatory arguments:

- `GET` requests take their arguments in the URL query (i.e. after a ?), in `application/x-www-form-urlencoded` format;
- other requests (`POST`, `PATCH`, etc.) take their arguments in the request body, in a format specified by the `Content-Type` header.

According to the HTTP standard, the default format for the HTTP request body is `application/x-www-form-urlencoded`; however the JSON format is much more suited to representing complex data types; as such, we recommend using `Content-Type: application/json` in request headers and JSON for the request body. The documentation will assume a JSON request in the examples provided.

### *3.1.1 HTTP persistent connections*

MollyBet API fully supports HTTP persistent connections as a way to reduce application response times by avoiding the creation of a separate HTTP connection for each new MollyBet API request. Please make sure your client supports HTTP persistent connections to make use of this functionality.

## 3.2 Synchronous Responses Format

### *3.2.1 HTTP Status Codes*

MollyBet API tries to return the appropriate HTTP status codes. These are only intended to be a general hint about the nature of the response being received; more details are contained within the response.

| Code | Meaning | Description |
|---|---|---|
| 200 | OK | Success! |
| 400 | Bad request | The request was invalid. It contained badly formatted data, or data that fails validation. For example: a required parameter is missing, or an order is placed with a negative price. |
| 401 | Unauthorised | Authentication credentials were missing from the request or incorrect. You need to re-authenticate and try again. |
| 403 | Forbidden | The request has been denied due to the current user not having sufficient permissions to perform the requested action. |
| 404 | Not found | The requested URI cannot be found on the server. This may refer to the specific URL, or to the resource being identified (e.g. a specific betslip). |
| 405 | Method not allowed | The given URI does not support this HTTP method (e.g. it supports `GET`, but you are trying to `POST`). |
| 406 | Not acceptable | The response format requested by the `Accept` HTTP header is not supported by the API. |
| 415 | Unsupported media type | The request is in a format specified by the `Content-Type` HTTP header that the API does not support. |
| 429 | Too many requests | You are being rate limited. |
| 500 | Internal Server Error | An unexpected error has occurred. |
| 503 | Service Unavailable | MollyBet API is unavailable or is being restarted. |

### *3.2.2 Envelope*

MollyBet API responses are wrapped in an envelope with a standard set of keys you can examine. A successful response will look like the following:

```
{
    "status": "ok",
    "data": "... some call-specific data here"
}
```

The `status` field will report `ok` for successful responses, or `error` when something has gone wrong. In addition to the `status` field, a successful response will always include a `data` field with the payload of the response.

Similarly for error responses, apart from the `status` field, they always contain a `code` field which corresponds to the name of the "exception" being raised. This is a machine- and human-readable string that describes the exact causes of the error encountered.

Furthermore, an error response may optionally contain a `data` field with a human-readable message giving more information about the error and how it may be resolved. For example, a 400 response to login could look like this:

```
{
    "status": "error",
    "code": "validation_error",
    "data": {
        "password": [
            "This field is required."
        ]
    }
}
```

### 3.2.3 Rate Limits

MollyBet API implements a rate limit policy to prevent abuse and ensure fair service to all the users of the service. The limits are generous and should not get in the way of regular use of our platform: they are mostly intended to prevent malicious use of the API and to guard against buggy programs. If you encounter HTTP response code 429 (Too many requests) during regular API usage please let us know.

## 3.3 Creating an Asynchronous Stream

A new asynchronous stream may be started by connecting a WebSocket to:

```
wss://api.mollybet.com/v1/stream/?token=TOKEN
```

where `TOKEN` is the session identifier returned by login.

As soon as a stream is connected, the client will automatically receive some initial state, which includes the current currency exchange rates, the connected customer's MollyBet balance, and a list of all the events we have offers for. This end of this operation is marked by the sync message.

Although we currently do not limit the number of WebSocket connections a client can do, it should be noted that that all connected WebSockets are completely independent from each other. Registering for offers on an event in one WebSocket causes the offers to be received only in that WebSocket, not in any other WebSockets. Different WebSockets can register to different events. Moreover, the `list_registered_events` command returns only the events registered in that WebSocket.

But keep in mind that, for the purpose of registered events limits checking, what counts is the aggregate number of registered across for a MollyBet group, regardless of how many users and how many WebSockets the event registrations are spread over.

Note: as explained in login, API sessions expire automatically after 24 hours of no activity, and websocket commands do not count as activity for this purpose. If you create a session just for websocket usage, you should send a normal HTTP request at least once every 24 hours.

## 3.4 Asynchronous Server Message Format

Much like synchronous responses, all the asynchronous messages are wrapped in a generic envelope. The envelope is a JSON object with fields:

`ts` **[*float*]**

The time when the message was transmitted through the async connection, as an Unix timestamp.

`data` **[*list[list[string, object], ...]*]**

The payload of the response, as a list of messages in the form of 2-items lists: [*code*, *payload*]. Possible codes and their matching payload types are described in Asynchronous Server Message Types.

```
{
    "ts": timestamp,
    "data": [
        [ code1, payload1 ],
        [ code2, payload2 ],
        ...
    ]
}
```

## 3.5 Asynchronous Client Message Format

Asynchronous messages sent from your software to MollyBet API take the form of a JSON list. You may send requests over the WebSocket connection to instruct MollyBet API to send one or more responses in Asynchronous Server Message Format to that message over that particular WebSocket connection.

The messages accepted by the WebSocket are in the form of a list whose first item is the message name and following items are the arguments supported by the command, if any. See Asynchronous Client Message Types for the list of supported messages.

For example to subscribe to offer updates for an event you would send:

```
["register_event", sport, event_id]
```

MollyBet API will respond with a response message informing about the success or failure of the command, and may include offer updates about the registered events in the asynchronous stream over the same WebSocket connection it received your client message.

## 3.6 Examples

**3.6.1 Login example** 10

**3.6.2 Synchronous request example** 11

**3.6.3 Asynchronous connection** 11

**3.6.4 Receiving event offers** 12

**3.6.5 Working with a betslip** 13

The following examples use HTTPie as an example of a HTTP client performing synchronous requests, and wscat as an example of WebSocket consumer. Any HTTP compliant client or library in any programming language will work the same way.

### *3.6.1 Login example*

A successful login can be performed by:

```
$ http POST https://api.mollybet.com/v1/sessions/ username=john \
    password=j0hnpass
```

Resulting in a request similar to:

```
POST /v1/sessions/ HTTP/1.1
Host: api.mollybet.com
Accept: application/json
Content-Type: application/json; charset=utf-8
```

```
{
  "username": "john",
  "password": "j0hnpass"
}
```

At which the API may return the following response, representing a successful login and reporting the session token to be used in the following requests:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "status": "ok",
  "data": "28f818ee4699ea482d0363c2262f5f5d",
}
```

### 3.6.2 *Synchronous request example*

Once authentication is successful the client can use the other available method to place orders and to query the current position, using the returned session token in the `Session` HTTP header. For example, the following command returns the open betslips:

```
$ http GET https://api.mollybet.com/v1/betslips/ \
    Session:28f818ee4699ea482d0363c2262f5f5d
```

Resulting in a request like:

```
GET /v1/betslips/ HTTP/1.1
Host: api.mollybet.com
Session: 28f818ee4699ea482d0363c2262f5f5d
```

And resulting in the following response (if no betslip is open):

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "status": "ok",
  "data": []
}
```

### 3.6.3 *Asynchronous connection*

The same token obtained in the login must be used to create a WebSocket connection:

```
$ wscat --connect \
    wss://api.mollybet.com/v1/stream/?token=28f818ee4699ea482d0363c2262f5f5d
```

which will immediately begin streaming:

```
< {
    "ts": 1477094004.385003,
    "data": [
      ["xrate", {"rate": 1.223, "ccy": "USD"}],
      ["xrate", {"rate": 1.1193, "ccy": "EUR"}],
      ["xrate", {"rate": 9.489, "ccy": "HKD"}],
      ["balance", {"open_stake": ["GBP", 0.0], "balance": ["GBP", 0.0]}],
      ["event", {"sport": "fb", "event_id": "2016-10-26,245,234",
        "home": "AC Chievo Verona", "away": "Bologna FC",
        "competition_id": "19", "competition_name": "Italy Serie A",
        "competition_country": "IT", "ir_status": "pre_event",
        "start_time": "2016-10-26T18:45:00+00:00"}],

      # a lot of other events...

      ["sync", {"token": "e55ab1d7edda4c31ac48283617d968df"}]
    ]
  }
```

### 3.6.4 Receiving event offers

If you are interested in the available offers or in other events happening to a sport event you can send a message to the WebSocket you are receiving messages from:

```
> ["register_event", "fb", "2016-10-26,245,234"]
```

Doing so the WebSocket will add messages to the stream with information about the offers available on the event as soon as they change, changes in the event status (when the match kicks off, when the score changes), and so on:

```
< {
    "ts": 1477326622.314035,
    "data": [
      ["response", {"status": "ok", "data": null}]
    ]
  }

< {
    "ts": 1477326622.328543,
    "data": [
      ["offer", {
        "bookie": "sing2", "market": "regular|fb-r-future|16^^",
        "selection": "2495514|r|70198|C|||",
        "sport": "fb", "event_id": "2016-10-26,245,234",
        "bet_type": "for,ah,a,-6", "in_running":false,
        "price_list": [{"bookie": {"price":2.09, "min": null, "max": null}}],
      }],
      ["offer", {
        "bookie": "bf-it", "market": "1.127996398", "selection": "12,back,0,0.0",
        "sport": "fb", "event_id": "2016-11-06,240,234",
        "bet_type": "for,cs,3,2", "in_running":false,
        "price_list":[
          {"bookie": {"price": 21.0, "min": ["GBP", 1.80], "max": ["GBP", 3.87]}},
          {"bookie": {"price":19.5, "min": ["GBP", 1.80], "max": ["GBP", 4.18]}}],
      }]
    ]
  }
```

```
< {
    "ts": 1477326625.239232,
    "data": [
      ["offer", {
        "bookie": "sing2", "market": "regular|fb-r-future|16^^",
        "selection": "2495514|r|70198|C|||",
        "sport": "fb", "event_id": "2016-10-26,245,234",
        "bet_type": "for,ah,a,-6", "in_running":false,
        "price_list": [{"bookie": {"price":2.10, "min": null, "max": null}}],
      }]
    ]
  }
```

### 3.6.5 *Working with a betslip*

In order to place an order a MollyBet API user must open a betslip using the Open Betslip call. Updates about the betslip state will be received on the asynchronous channel. For example, in order to bet on the above `event`:

```
$ http POST https://api.mollybet.com/v1/betslips/ \
    Session:28f818ee4699ea482d0363c2262f5f5d \
    sport=fb event_id=2016-10-26,245,234 bet_type=for,h
```

The command will result both in a synchronous reply:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "status": "ok",
  "data": {
    "accounts": [
      {"bet_type": "for,h", "bookie": "bf", "username": "_40a78ed8_"},
      {"bet_type": "for,ah,h,-2", "bookie": "pin", "username": "_6d241fed_"},
    ],
    "sport": "fb",
    "bet_type": "for,h",
    "betslip_id": "b2e936565fea4d778c38269ab1667f17",
    "bookies_with_offers": ["ibc", "sing2"],
    "close_reason": "still open",
    "customer_ccy": "eur",
    "customer_username": "john",
    "equivalent_bets": true,
    "equivalent_bets_bookies": [],
    "event_id": "2016-10-26,245,234",
    "expiry_ts": 1477326778.358509,
    "is_open": true,
    "multiple_accounts": false,
    "obfuscate_accounts": false,
    "obfuscate_bookies": false
  }
}
```

but also in a sequence of messages on the asynchronous stream, updating about the state of the betslip:

```
< {
    "ts": 1477326723.482099,
    "data": [
      ["betslip", { ... same data of the sync channel }]
    ]
  }

< {
    "ts": 1477326723.584029,
    "data": [
      ["pmm", {
        "betslip_id": "b2e936565fea4d778c38269ab1667f17",
        "sport": "fb",
        "event_id": "2016-10-26,245,234",
        "bookie": "bf",
        "username": "_40a78ed8_",
        "bet_type": "for,h",
        "status": {"code": "success"},
        "price_list": [
          {
            "bookie": {"price": 2.16, "max": ["GBP", 5.4], "min": ["GBP", 2.0]},
            "effective": {"price": 2.1368, "max": ["GBP", 5.4], "min": ["GBP", 2.0]}
          },
          {
            "bookie": {"price": 2.12, "min": ["GBP", 2.0], "max": ["GBP", 278.57]},
            "effective": {"price": 2.0976, "min": ["GBP", 2.0], "max": ["GBP", 278.57]}
          }]
      }]
    ]
  }

< {
    "ts": 1477326723.796697,
    "data": [
      ["pmm", {
        "betslip_id": "b2e936565fea4d778c38269ab1667f17",
        "sport": "fb",
        "event_id": "2016-10-26,245,234",
        "bookie": "pin",
        "username": "_3aa3dcac_",
        "bet_type": "for,h",
        "status": {"code": "success"},
        "price_list": [
          {
            "bookie": {"max": ["EUR", 1838.4], "price": 2.09, "min": ["EUR", 45.96]},
            "effective": {"max": ["GBP", 1640.11], "price": 2.09, "min": ["GBP", 41.00]}
          }]
      }]
    ]
  }

< {
    "ts": 1477326723.958774,
    "data": [
      ["pmm",
        ...continuous stream of PMM updates until the betslip is closed
```

```
    ]
  }

< {
    "ts": 1477326733.137329,
    "data": [
      ["betslip_closed", {
        "close_reason": "explicit close request",
        "betslip_id": "b2e936565fea4d778c38269ab1667f17"
      }]
    ]
  }
```

# 4 Available Requests

Users who have been granted access to MollyBet API can log in using the same credentials they use to access the website.

A successful login request will return a *session token* which **must** be provided in the `Session` HTTP header when making subsequent requests (and can be reused for multiple requests). The same identifier is also required to initiate an asynchronous stream.

**4.1 Sessions Requests** **17**

4.1.1 Login and create a new MollyBet API session 17

4.1.2 Return metadata associated with a session 18

4.1.3 Logout and terminate a session 18

4.1.4 Create a new session to impersonate a different user 18

4.1.5 Create a new session for one of the users visible from the current session, while checking the validity of the credentials provided 19

**4.2 Betslips Requests** **19**

4.2.1 Open a new betslip for a given event and bet type 20

4.2.2 List the currently open betslips 22

4.2.3 Get the current state of an open betslip 23

4.2.4 Extend the betslip expiration timeout 24

4.2.5 Close a betslip 24

**4.3 Orders Requests** **24**

4.3.1 Create a new bet placement order 25

4.3.2 Return up-to-date information about an order 28

4.3.3 Close immediately an order 30

4.3.4 Return information about placed orders matching certain requisites 30

4.3.5 Return the list of filter options for placers 32

4.3.6 Return the list of filter options for countries 32

4.3.7 Return the list of filter options for competitions 32

4.3.8 Return the list of filter options for events 33

4.3.9 Calculate a position for placed orders 33

4.3.10 Return the order processing log 34

**4.4 Positions Requests** **36**

4.4.1 Return information about a bet type, including win/loss grid 36

**4.5 Customers Requests** **37**

4.5.1 Create a new MollyBet account 37

4.5.2 List all the MollyBet accounts known by the requesting user 38

4.5.3 Return information about a single MollyBet account 38

4.5.4 Change some of the attributes of a MollyBet account 39

4.5.5 Check if the current user can place bets with this IP 39

4.5.6 Return the list of the bookmaker accounts available to the customer 39

4.5.7 Get the accounting summary of a MollyBet account 40
4.5.8 Return the history of the credit limit changes of a MollyBet account 40
4.5.9 Request a refresh of the balance for a customer's bookie account 41
4.5.10 Get a provisioning key for two-step authentication 41
4.5.11 Check if a customer has enabled two-step authentication 42
4.5.12 Enable or disable two-step authentication for a customer 42
**4.6 Groups Requests 43**
4.6.1 Return the list of the known MollyBet account groups 43
4.6.2 Return information about a MollyBet group 43
4.6.3 Return the list of the bookmakers accounts available to a group 43
4.6.4 Set the new credit limit for a group 44
4.6.5 Return the history of the credit limit changes of a MollyBet group 45
**4.7 Transfers Requests 45**
4.7.1 Register a new transfer between the agent and a customer 45
4.7.2 Return the history of transfers performed by an agent 45
**4.8 Miscellaneous Requests 46**
4.8.1 Return the list of exchange rates between GBP and other currencies 46

## 4.1 Sessions Requests

These API calls allow the creation and destruction of sessions used to interact with MollyBet API. login creates a new session, logout terminates a session.

**4.1.1 Login and create a new MollyBet API session 17**
**4.1.2 Return metadata associated with a session 18**
**4.1.3 Logout and terminate a session 18**
**4.1.4 Create a new session to impersonate a different user 18**
**4.1.5 Create a new session for one of the users visible from the current session, while checking the validity of the credentials provided 19**

### *4.1.1 Login and create a new MollyBet API session*

**Method and URL:**

```
POST /v1/sessions/
```

The *session ID* returned by this request must be used as the authentication token for further API requests, by setting it as the value of the `Session` HTTP header.

In order to terminate the session use the logout call.

Note: API sessions will expire automatically after 24 hours without any activity. If you wish to keep an API session token active, you should send a valid request, (e.g. `GET /v1/xrates/`) at least once every 24 hours. Be aware that websocket-only activity does not count towards keeping a session active.

**Request Parameters:**

`username` **[ *string* ]:**

MollyBet account username

`password` **[ *string* ]:**

MollyBet account password

`token` **[ *string* ] (optional):**

A valid authenticator token or recovery code. This is only necessary for customers who have enabled two-step authentication.

If you do not know in advance whether the customer has two-step authentication enabled, try first without a token. If you receive error code `token_required_but_not_provided`, prompt the customer for a token and repeat the request.

**Return Data:** a string with the new *session ID* if authentication succeeds

### *4.1.2 Return metadata associated with a session*

**Method and URL:**

```
GET /v1/sessions/<session_id>/
```

**Example output data:**

```
{
    "username": "johndoe",
    "client_type": "direct"
}
```

**URL Parameters:**

`session_id`:

The id of the session to obtain information for

### *4.1.3 Logout and terminate a session*

**Method and URL:**

```
DELETE /v1/sessions/<session_id>/
```

After a successful logout, the session id can no longer be used as an authentication token, and you will need to generate a fresh one with a new call to login.

**URL Parameters:**

`session_id`:

The id of the session to terminate

### *4.1.4 Create a new session to impersonate a different user*

**Method and URL:**

```
POST /v1/sessions/<session_id>/sudo/
```

Note that the original session is not deleted or changed in any way. You can start making new requests using the new session ID. To "unsudo" simply logout from the session created by this call and go back to use the original *session ID*.

**URL Parameters:**

`session_id`:

The session id of the active user

**Request Parameters:**

`target` **[ *string* ]:**

MollyBet username of the user to impersonate

**Return Data:** A new *session ID*, as for the login call

### *4.1.5 Create a new session for one of the users visible from the current session, while checking the validity of the credentials provided*

**Method and URL:**

```
POST /v1/sessions/<session_id>/sublogin/
```

The new session has its privileges restricted to only what would normally be accessible through the website.

Note that the original session is not deleted or changed in any way. You can start making new requests using the new session ID. To "unsudo" simply logout from the session created by this call and go back to use the original *session ID*.

**URL Parameters:**

`session_id`**:**

The session id of the active user

**Request Parameters:**

`target` **[ *string* ]:**

The target user's username

`password` **[ *string* ]:**

The target user's password

`token` **[ *string* ] (optional):**

A valid authenticator token or recovery code. This is only necessary for customers who have enabled two-step authentication.

If you do not know in advance whether the customer has two-step authentication enabled, try first without a token. If you receive error code `token_required_but_not_provided`, prompt the customer for a token and repeat the request. Alternatively, you can query the customer's two-step auth authentication status before you make the login request.

**Return Data:** A new *session ID*, as for the login call

## 4.2 Betslips Requests

A betslip must be open when an API client wants to place bets on a certain event.

The betslip will select the best available accounts to use for placing bets on the selected bet type, and will commence a *PMM* cycle on each of those accounts. A *PMM* (*Price, Min and Max*) is the process of repeatedly opening up a betslip on the bookie website to ensure we have the latest price for that bet type, and also the minimum and maximum stakes available for that bet.

The state of the betslips is sent to the client via the asynchronous stream, using async betslips messages.

**4.2.1 Open a new betslip for a given event and bet type** 20

**4.2.2 List the currently open betslips** 22

**4.2.3 Get the current state of an open betslip** 23

**4.2.4 Extend the betslip expiration timeout** 24

**4.2.5 Close a betslip** 24

### 4.2.1 *Open a new betslip for a given event and bet type*

**Method and URL:**

```
POST /v1/betslips/
```

When a betslip is open the system will fetch the available prices and stake ranges on the supplied event (as determined by event_id). The betslip is identified by a `betslip_id` attribute which should be used to manipulate its state in further MollyBet API calls.

**Request Parameters:**

`sport` **[ *string* ]:**

The sport for the event in which you're interested

`event_id` **[ *string* ]:**

The event ID for the event in which you're interested

`bet_type` **[ *string* ]:**

The bet type in which you're interested

`equivalent_bets` **[ *bool* ] (optional, default:** `true`**):**

Use bets that are equivalent to the bet type you've requested in addition to the bet type you've actually asked for.

For example, a bet on under 0.5 goals is equivalent to a bet on correct score 0-0. With this option set, we'll try and bet on both (assuming there are sufficient bookie accounts available).

`multiple_accounts` **[ *bool* ] (optional, default:** `false`**):**

If you have multiple accounts with a given bookie, add all of them to the betslip.

This means that if this betslip is used for placing an order then we may potentially place multiple bets on the same bookie simultaneously. Note, however, that with this option set equivalent bets will only be used if the requested bet type is not available.

If this option is not given, then a single account per bookie will be selected and used.

`want_bookies` **[ *list[string, ...]* ] (optional):**

The set of bookies from which to select accounts to display on the betslip. If this option is omitted, then accounts from all available bookies will be used.

`equivalent_bets_bookies` **[ *list[string, ...]* ] (optional):**

List of bookies for which equivalent bet types should be enabled.

If omitted or empty, equivalent bet types will apply to all bookies.

`wait_for_pmms` **[ *float* ] (optional, maximum:** `5`**):**

Wait for a certain period for the PMMs to arrive.

By default, the betslip is returned immediately without waiting for PMMs. Setting this option to a float instructs MollyBet to wait up to a maximum of this number of seconds for PMMs to arrive for all accounts selected for the betslip.

**Return Data:** An object with the new betslip attributes.

**Output Format:**

`betslip_id` **[ *string* ]:**

The unique identifier for the betslip.

`sport` **[ *string* ]:**

The sport for the event.

`event_id` **[ *string* ]:**

The event ID for the event.

`bet_type` **[ *string* ]:**
: The bet type of the betslip.

`equivalent_bets` **[ *bool* ]:**
: Whether equivalent bets are enabled for this betslip.

`multiple_accounts` **[ *bool* ]:**
: Whether multiple accounts are enabled for this betslip.

`want_bookies` **[ *list[string, ...]* ]:**
: The set of bookies enabled for this betslip.

`is_open` **[ *bool* ]:**
: If `true` this betslip is still open.

`expiry_ts` **[ *float* ]:**
: Unix timestamp for the time when the betslip will expire.

`bookies_with_offers` **[ *list[string, ...]* ]:**
: List of bookies which have offers for this selection.

`equivalent_bets_bookies` **[ *list[string, ...]* ]:**
: The `equivalent_bets_bookies` as provided by the user.

`close_reason` **[ *string* ]:**
: The reason, if any, why the betslip was closed.

`customer_username` **[ *string* ]:**
: The username for the account that opened the betslip.

`customer_ccy` **[ *string* ]:**
: The currency code for the account that opened the betslip.

`invalid_accounts` **[ *object{string -> object{string -> string}}* ]:**
: Any invalid accounts that were found, with the reason.

`accounts` **[ *list[object, ...]* ]:**
: list of accounts currently selected for this betslip.

The `accounts` attribute is a list of objects with format:

`bookie` **[ *string* ]:**
: The bookmaker code of this account.

`username` **[ *string* ]:**
: The username of this accout.

`bet_type` **[ *string* ]:**
: A string representing the bet type.

  See List of Bet Types for explanations.

`status` **[ *string* ] (optional):**
: Status of the last pmm on this account, if any.

`price_list` **[ *list[object, ...]* ] (optional):**
: Price list of the last PMM on this account, if any.

  Every object of the list has two attribute:

  `bookie` **[ *list[object, ...]* ]:**
  : The original price and min/max entries as received by the bookie.

  `effective` **[ *list[object, ...]* ]:**
  : The net price and min/max after taking commission, referral fees, etc. into account.

  Both lists contain PMM objects with attributes:

`price` **[ *float* ]:**

The decimal price of the offer.

`min` **[ *list[string, float]* ]**

The minimum stake that can be placed as a 2-items list: [*currency code*, *amount*].

`max` **[ *list[string, float]* ]**

The maximum stake that can be placed as a 2-items list: [*currency code*, *amount*].

Normal bookies only offer one PMM for their selections, so their price list will contain a single item. Exchanges may have different prices for each selection and the list will report all of them.

See the Get Betslip output for an example of these objects.

**Example:**

**Example request:**

```
{
    "sport": "fb",
    "event_id": "2016-10-17,49,53",
    "bet_type": "for,h"
}
```

**Example return data:**

This is a sample of a betslip returned by the Open Betslip call, so it is returned before the bookie can report the current price and available stake: these details can be returned later (see the Get Betslip output for example) or via asynchronous messages.

```
{
  "betslip_id": "cd75b859154f44b28e7dbd637c5b92fb",
  "sport": "fb",
  "event_id": "2016-10-17,49,53",
  "bet_type": "for,h",
  "equivalent_bets": true,
  "multiple_accounts": false,
  "is_open": true,
  "expiry_ts": 1476197232.96185,
  "bookies_with_offers": ["bdaq", "bf", "sing2", "pin", "ibc", "mbook", "sbo"],
  "equivalent_bets_bookies": [],
  "close_reason": "still open",
  "customer_username": "johndoe",
  "customer_ccy": "eur",
  "invalid_accounts": {},
  "accounts": [
    {"bookie": "bdaq", "username": "_a6649ca0_", "bet_type": "against,ah,a,-2"},
    {"bookie": "bdaq", "username": "_a6649ca0_", "bet_type": "for,ah,h,-2"},
    {"bookie": "bdaq", "username": "_a6649ca0_", "bet_type": "for,h"},
    {"bookie": "ibc", "username": "_de96a92f_", "bet_type": "for,ah,h,-2"},
    {"bookie": "ibc", "username": "_62ce0e10_", "bet_type": "for,h"},
    {"bookie": "sbo", "username": "_fd3fed83_", "bet_type": "for,ah,h,-2"},
    {"bookie": "sbo", "username": "_056aa2f7_", "bet_type": "for,h"}
  ]
}
```

### *4.2.2 List the currently open betslips*

**Method and URL:**

```
GET /v1/betslips/
```

**Return Data:** A list containing the *betslip_id* of each currently open betslip.

### 4.2.3 Get the current state of an open betslip

**Method and URL:**

```
GET /v1/betslips/<betslip_id>/
```

The betslip state includes the same attributes returned by Open Betslip and the state that has been accumulated asynchronously with the PMM responses. Note that the most efficient way to react to PMMs is not to use this call but to listen for PMMs on the async channel.

**URL Parameters:**

`betslip_id`:

The identifier of the betslip as returned by Open Betslip.

**Return Data:** An object with the same format of the Open Betslip output, possibly with more information.

**Example:**

**Example output:**

```
{
  "betslip_id": "923264e50fe34ab49053d2a369a3a7be",
  "sport": "fb",
  "event_id": "2016-10-17,49,53",
  "bet_type": "for,h",
  "bet_type_description": "HOME",
  "bet_type_template": "{home}",
  "equivalent_bets": true,
  "multiple_accounts": false,
  "is_open": true,
  "expiry_ts": 1476199028.620634,
  "bookies_with_offers": ["bdaq", "bf", "sing2", "pin", "ibc", "mbook", "sbo"],
  "equivalent_bets_bookies": [],
  "close_reason": "still open",
  "customer_username": "johndoe",
  "customer_ccy": "eur",
  "invalid_accounts": {},
  "accounts": [
    {
      "bookie": "bdaq", "username": "_a6649ca0_", "bet_type": "against,ah,a,-2",
      "status": "success", "price_list": [
        {
          "bookie": {
            "price": 2.075, "min": ["GBP", 0.46], "max": ["GBP", 999.75]
          },
          "effective": {
            "price": 2.064, "min": ["GBP", 0.46], "max": ["GBP", 999.75]
          }
        },
        {
          "bookie": {
            "price": 2.0, "min": ["GBP", 0.5], "max": ["GBP", 1000.0]
          },
          "effective": {
            "price": 1.99, "min": ["GBP", 0.5], "max": ["GBP", 1000.0]
          }
        }
      ]
    },
    {
      "bookie": "sbo", "username": "_056aa2f7_", "bet_type": "for,h",
      "status": "success", "price_list": [
        {
          "bookie": {
            "price": 2.12, "min": ["MYR", 4.5], "max": ["MYR", 6923.7]
          },
          "effective": {
            "price": 2.12, "min": ["GBP", 0.81], "max": ["GBP", 1356.01]
          }
        }
      ]
    }
  ]
}
```

### 4.2.4 *Extend the betslip expiration timeout*

**Method and URL:**

```
POST /v1/betslips/<betslip_id>/refresh/
```

Betslips have an expiration timeout (see the `expiry_ts` attribute in Get Betslip). If you want to keep the betslip open for longer you can use this method.

**URL Parameters:**

`betslip_id`:

The identifier of the betslip as returned by Open Betslip.

### 4.2.5 *Close a betslip*

**Method and URL:**

```
DELETE /v1/betslips/<betslip_id>/
```

Closing the betslip releases the resources associated to it, in particular it stops the PMM cycle.

**URL Parameters:**

`betslip_id`:

The identifier of the betslip as returned by Open Betslip.

## 4.3 Orders Requests

These functions are used to place new orders on MollyBet and to retrieve and manipulate the state of placed orders.

**4.3.1 Create a new bet placement order** 25

**4.3.2 Return up-to-date information about an order** 28

**4.3.3 Close immediately an order** 30

4.3.4 Return information about placed orders matching certain requisites 30
4.3.5 Return the list of filter options for placers 32
4.3.6 Return the list of filter options for countries 32
4.3.7 Return the list of filter options for competitions 32
4.3.8 Return the list of filter options for events 33
4.3.9 Calculate a position for placed orders 33
4.3.10 Return the order processing log 34

### *4.3.1 Create a new bet placement order*

**Method and URL:**

```
POST /v1/orders/
```

When an order is created, MollyBet API will attempt to place the `stake` requested, at a price equal or better than the requested `price`. MollyBet will select the optimal account, placing as much stake as possible on the bookies offering the best price, automatically performing conversion in different currencies if needed. All the accounts available will be considered, unless limited by the `accounts` parameter.

Placement will happen immediately provided that the price and stake requested are available on the market. If not, the order will remain in open state until new suitable offers appear, or the timeout specified in `duration` expires.

Note: MollyBet API will often not place full stake due to rounding. Typically, bookie accounts have constraints on the stake amount that they accept. In some cases, the stake amount must be a multiple of a certain value (stake increment), and have a minimum value, and possibly a maximum value. If, after conversion of the stake from the requested order currency into the bookie account currency, followed by rounding down to the nearest multiple of the stake increment, we find that the stake is below the minimum stake for that bookie account, then MollyBet API will not attempt to place in that account.

Note also: unless the `no_put_offer_exchange` option is set to `true`, MollyBet API will attempt to put an offer on an exchange if it cannot find any liquidity to take to fulfill the order stake.

**Request Parameters:**

`betslip_id` **[ *string* ]:**

The ID of an open betslip, from which the *sport*, *event_id*, and *bet_type* will be taken.

`price` **[ *float* ] (minimum:** `1.0`**, maximum:** `1000`**):**

The minimum price at which you're happy to bet.

`stake` **[ *list[string, float]* ]:**

The amount you wish to stake on this bet (a `[ccy_code, amount]` list)

`duration` **[ *float* ] (optional, default:** `15`**, minimum:** `1`**, maximum:** `259200`**):**

The length of time, in seconds, for which you want MollyBet to try and place the order.

`keep_open_ir` **[ *bool* ] (optional, default:** `false`**):**

If possible, keep the order open until the first goal or red card (rather than closing it as soon as the event goes in-running). The order will still be closed if it is filled or expires beforehand.

`accounts` **[ *list[list[string, string], ...]* ] (optional, default:** `[]`**):**

Restrict placement to the specified list of bookie accounts. Otherwise, placement is allowed to use all accounts.

Accounts are specified as a list of 2-item lists [*bookie*, *username*], e.g.:

```
[["bdaq", "_a6649ca0_"], ["sbo", "_056aa2f7_"]]
```

Note: orders normally are restricted to place on the accounts that were selected into the betslip at the time the order was created. Only if the `adaptive_bookies` option is given are new accounts potentially used for the order.

Tip: to retrict placement to certain bookies, rather than passing all accounts of that bookie in this option it is better to: (1) open the betslip with the `want_bookies` option, then (2) omit the `accounts` option (this one) of the create order request.

`adaptive_bookies` **[ *list[string, ...]* ] (optional):**

The set of bookies from which MollyBet is allowed to re-select accounts in case of bookie errors during placement. Special cases: (1) if the parameter is omitted or null (the default), account re-selection is disabled, (2) if the parameter value is an empty list, re-selection is enabled for all bookies.

`ignore_system_maintenance` **[ *bool* ] (optional, default:** `false`**):**

Place the order even when the system is reported under maintenance. Use this option at your own risk.

`no_put_offer_exchange` **[ *bool* ] (optional, default:** `false`**):**

If `true`, placement will never attempt to put an order up in an exchange when no liquidity is found in regular bookies

`bookie_min_stakes` **[ *object{string -> list[string, float]}* ] (optional, default:** `{}`**):**

Only place a bet with certain bookies if intending to request at least a certain stake. Format is an object with *bookie codes* as keys and 2-items lists [*currency code*, *amount*] as value, e.g.:

```
{
  "bf": ["GBP", 50.00],
  "sbo": ["HKD", 1000.00]
}
```

`user_data` **[ *string* ] (optional):**

Extra user data to attach to the order. It will be reported back as the *user_data* field in orders json. Max length: 256 unicode characters.

`request_uuid` **[ *UUID string* ] (optional):**

A UUID that uniquely identifies this request.

This is used to prevent duplicate orders from being placed in the event that your request is repeated. If you try to create multiple orders using the same UUID within a 24-hour period, only the first request will be acted upon.

**Return Data:** A structure containing details of the order placed.

**Output Format:**

`order_id` **[ *int* ]:**

The unique ID of the order.

`bet_type` **[ *string* ]:**

The bet type of the order.

`bet_type_description` **[ *string* ]:**

An human-readable explanation of the bet type.

`sport` **[ *string* ]:**

The sport of the order.

`placer` **[ *string* ]:**

MollyBet account username that placed the order.

`want_price` **[ *float* ]:**

The minimum price requested for the order.

`want_stake` **[ *list[string, float]* ]:**
: The total stake requested to place, in [*currency code*, *amount*] format.

`ccy_rate` **[ *float* ]:**
: The exchange rate from Pounds Sterling to the want currency at the time of placement, i.e.

  *amount in want currency = amount in GBP × ccy_rate*.

`placement_time` **[ *ISO 8601 datetime string* ]:**
: The date and time at which the order was placed.

`expiry_time` **[ *ISO 8601 datetime string* ]:**
: The date and time at which the order expires.

`closed` **[ *bool* ]:**
: `true` if the order is closed.

`close_reason` **[ *string* ]:**
: If the order is closed, a code reporting why the error was closed (e.g. `order_filled`, `timed_out`, `cancelled`, etc.)

`event_info` **[ *object* ]:**
: A structure reporting known information about the sport event the order was placed on.

`bets` **[ *list[object, ...]* ] (optional):**
: The list of bets placed to fill this order.

  This field is missing at placement time because placement happens asynchronously; further calls to query the order state (e.g. Order Info, Orders List...) will contain a list of bets. The description for this structure is in the Order Info format documentation.

`user_data` **[ *string* ]:**
: Whatever value was provided as the `user_data` field in the `POST /v1/orders` request, if any.

`status` **[ *string* ]:**
: The current status of the order. See list of order statuses.

The `event_info` structure contains the following attributes:

`event_id` **[ *string* ]:**
: An opaque identifier for the event.

`home_id` **[ *int* ]:**
: The unique ID of the home team.

`home_team` **[ *string* ]:**
: The name of the home team.

`away_id` **[ *int* ]:**
: The unique ID of the away team.

`away_team` **[ *string* ]:**
: The name of the away team.

`competition_id` **[ *int* ]:**
: The unique ID of the event competition.

`competition_name` **[ *string* ]:**
: The name of the event competition.

`start_time` **[ *ISO 8601 datetime string* ]:**
: The start time of the event, if known.

`date` **[ *ISO 8601 date string* ]:**
: The date of the event.

Note that this date usually refers to European time: different bookmakers in different time zones may report an event under different dates.

**Example:**

An example result for a order placement response could be:

```
{
  "order_id": 19581948,
  "bet_type": "for,ahunder,40",
  "bet_type_description": "Corners: Under 10.0 corners (Asian)",
  "sport": "fb_corn",
  "placer": "johndoe",
  "want_price": 2.0,
  "want_stake": ["EUR", 60.0],
  "ccy_rate": 1.160200,
  "placement_time": "2016-09-27T14:16:11.719478+00:00",
  "expiry_time": "2016-09-27T14:17:11.718960+00:00",
  "closed": false,
  "close_reason": null,
  "event_info": {
    "event_id": "2016-09-27,194,178",
    "home_id": 194,
    "home_team": "BV Borussia 09 Dortmund",
    "away_id": 178,
    "away_team": "Real Madrid CF",
    "competition_id": 28,
    "competition_name": "UEFA Champions League",
    "competition_country": "EU",
    "start_time": "2016-09-27T18:45:00+00:00",
    "date": "2016-09-27",
  }
}
```

### *4.3.2 Return up-to-date information about an order*

**Method and URL:**

```
GET /v1/orders/<order_id>/
```

**URL Parameters:**

`order_id`**:**

The unique ID of the order as returned by Open Order.

**Return Data:** A structure containing info about the requested order.

**Output Format:**

The structure returned by the call is the same of the one returned by Open Order, but it will contain the list of placed bets too.

The format of the bet structure is the following:

`bet_id` **[ *int* ]:**

The unique ID of the bet.

`order_id` **[ *int* ]:**

The unique ID of the order the bet was placed for.

`bookie_bet_id` **[ *string* ]:**

The bet ID returned by the bookmaker.

`status` **[ *object* ]:**

The current state of the bet. A JSON structure containing the attributes:

`code` **[ *string* ]:**

The code of the status, see list of bet statuses.

`reason` **[ *string* ] (optional):**

The reason for the bet state, usually available only for failed bets.

`sport` **[ *string* ]:**

The sport of the order.

`event_id` **[ *string* ]:**

An opaque identifier for the event.

`bookie` **[ *string* ]:**

The code of the bookmaker the bet was placed on.

`username` **[ *string* ]:**

The bookmaker account username the bet was placed on.

`bet_type` **[ *string* ]:**

The bet type of the order.

`ccy_rate` **[ *float* ]:**

The exchange rate from Pounds Sterling to the bet currency at the time of placement, i.e.

*amount in bet currency* = *amount in GBP* × *ccy_rate*.

`want_price` **[ *float* ]:**

The price requested for this bet.

`got_price` **[ *float* ]:**

The price obtained by this bet.

`want_stake` **[ *list[string, float]* ]:**

The stake requested to place for this bet, in [*currency code*, *amount*] format.

`got_stake` **[ *list[string, float]* ]:**

The stake placed in this bet, in [*currency code*, *amount*] format.

`profit_loss` **[ *list[string, float]* ] (optional):**

The profit or loss of this bet, if settled, in [*currency code*, *amount*] format.

`reconciled` **[ *bool* ]:**

*true* if MollyBet and the bookmaker agree on the bet profit loss, *false* if there is a disagreement (while it is investigated). *null* if the bet is still unsettled.

**Example:**

The same order in the Open Order example may be returned as the following structure after one bet is placed and settled:

```
{
  "order_id": 19581948,
  "bet_type": "for,ahunder,40",
  "bet_type_description": "Corners: Under 10.0 corners (Asian)",
  "sport": "fb_corn",
  "placer": "johndoe",
  "want_price": 2.0,
  "want_stake": ["EUR", 60.0],
  "ccy_rate": 1.160200,
  "placement_time": "2016-09-27T14:16:11.719478+00:00",
  "expiry_time": "2016-09-27T14:17:11.718960+00:00",
  "closed": true,
  "close_reason": "order_filled",
  "event_info": {
    "event_id": "2016-09-27,194,178",
    "home_id": 194,
    "home_team": "BV Borussia 09 Dortmund",
    "away_id": 178,
    "away_team": "Real Madrid CF",
    "competition_id": 28,
    "competition_name": "UEFA Champions League",
    "competition_country": "EU",
    "start_time": "2016-09-27T18:45:00+00:00",
    "date": "2016-09-27",
  },
  "bets": [
    {
      "bet_id": 3324372947,
      "order_id": 19581948,
      "bookie_bet_id": "5575992964",
      "status": {
        "code": "settled"
      },
      "sport": "fb_corn",
      "event_id": "2016-09-27,194,178",
      "bookie": "sbo",
      "username": "_ae3d7f51_",
      "bet_type": "for,ahunder,40",
      "ccy_rate": 10.068,
      "want_price": 2.01,
      "got_price": 2.010101,
      "want_stake": ["HKD", 502.52],
      "got_stake": ["HKD", 502.52],
      "reconciled": true,
      "profit_loss": ["HKD", -502.52],
    }
  ]
}
```

### *4.3.3 Close immediately an order*

**Method and URL:**

```
POST /v1/orders/<order_id>/close/
```

MollyBet will stop placing new bets to fulfill this order, without waiting for the timeout to expire. This may not cancel bets that are already being placed.

**URL Parameters:**

`order_id`:

The ID of the order to cancel.

### *4.3.4 Return information about placed orders matching certain requisites*

**Method and URL:**

```
GET /v1/orders/
```

**Request Parameters:**

`page` **[ *int* ] (optional, default:** `1`**, minimum:** `1`**):**

Number of the page to retrieve.

`page_size` **[ *int* ] (optional, default:** `25`**, minimum:** `1`**, maximum:** `1000`**):**

Maximum number of items per page.

`order_by` **[ *list[string, ...]* ] (optional, default:** `["order_id desc"]`**):**

How to order the results, as a list of "*attribute* `asc`|`desc`" strings.

`placer` **[ *string* ] (optional):**

Show only orders placed by this MollyBet account (username).

`group_id` **[ *int* ] (optional):**

Show only orders placed by accounts in the given group.

`date_from` **[ *ISO 8601 datetime string* ] (optional):**

Show only orders placed from this date/time on.

`date_to` **[ *ISO 8601 datetime string* ] (optional):**

Show only orders placed before this date/time.

`event_date_from` **[ *ISO 8601 date string* ] (optional):**

Show only orders on event played from this date on.

`event_date_to` **[ *ISO 8601 date string* ] (optional):**

Show only orders on event played before this date.

`event_start_from` **[ *ISO 8601 datetime string* ] (optional):**

Show only orders on events that start after this date/time. Note: in some rare cases only the date of the event is known, not the start time. In such cases, the start time is assumed to be 12:00 UTC for the purpose of this filter.

`event_start_to` **[ *ISO 8601 datetime string* ] (optional):**

Show only orders on events that start before this date/time. Note: same note from event_start_from applies here.

`is_open` **[ *bool* ] (optional):**

If value is 'true', show only orders open or containing unsettled bets. If 'false', show only orders that are closed and contain only settled bets. Failed orders are omitted in either case unless the is_open filter is not used at all.

`sport` **[ *string* ] (optional):**

Show only orders placed on this sport (a sport code).

`event_id` **[ *list[string, ...]* ] (optional, default:** `[]`**):**

Show only orders placed on this event.

`competition_id` **[ *int* ] (optional):**

Show only orders placed on this competition (specified by competition id).

`competition_country` **[ *string* ] (optional):**

Show only orders placed on this country (a two-letters country code).

`search` **[ *string* ] (optional):**

Show only orders with a certain substring in the players/teams names.

`placed_inrunning` **[ *bool* ] (optional):**

If value is 'true', include only orders placed after the event has started. If value is 'false', only orders placed before the event started are included. If ommitted or null, both inrunning and deadball orders are included.

`status` **[ *string* ] (optional):**

Show only orders with selected statuses.

`wanted_ids` **[ *list[string, ...]* ] (optional):**

Filter orders that contain these IDs. These might be present as order IDs, bet IDs or bookie bet IDs.

**Return Data:** A list of order structures matching the filters provided in the request.

### *4.3.5 Return the list of filter options for placers*

**Method and URL:**

```
GET /v1/orders/filters/placers/
```

**Output Format:**

A list of objects of the form:

`username` **[ *string* ]:**

The unique username of the placer, to be used for filtering.

`name` **[ *string* ]:**

The display name for the placer option.

### *4.3.6 Return the list of filter options for countries*

**Method and URL:**

```
GET /v1/orders/filters/countries/
```

**Output Format:**

A list of objects of the form:

`competition_country` **[ *string* ]:**

The unique ID of the country, to be used for filtering.

`name` **[ *string* ]:**

The display name for the country option.

### *4.3.7 Return the list of filter options for competitions*

**Method and URL:**

```
GET /v1/orders/filters/competitions/
```

**Request Parameters:**

`competition_country` **[ *string* ] (optional):**

The unique ID of the event country.

**Output Format:**

A list of objects of the form:

`competition_id` **[ *integer* ]:**

The unique ID of the competition, to be used for filtering.

`competition_country` **[ *string* ]:**

The unique ID of the competition country.

`name` **[ *string* ]:**

The display name for the competition option.

### 4.3.8 *Return the list of filter options for events*

**Method and URL:**

```
GET /v1/orders/filters/events/
```

**Request Parameters:**

`competition_country` **[ *string* ] (optional):**

The unique ID of the event country.

`competition_id` **[ *int* ] (optional):**

The unique ID of the event competition.

`event_date_from` **[ *ISO 8601 date string* ] (optional):**

Only return events on or after this date.

`event_date_to` **[ *ISO 8601 date string* ] (optional):**

Only return events before or on this date.

**Output Format:**

A list of objects of the form:

`event_id` **[ *string* ]:**

The unique ID of the event, to be used for filtering.

`name` **[ *string* ]:**

The display name for the event option.

### 4.3.9 *Calculate a position for placed orders*

**Method and URL:**

```
GET /v1/orders/position/
```

**Request Parameters:**

The method accepts the same parameters of the Orders List call.

**Output Format:**

`payoff_grid` **[ *object{*`ccy_code` *-> string,* `values` *-> list[list[float, ...], ...]}* ]:**

An object reporting what would be the Profit/Loss according to each possible outcome of the sport event. It contains a currency code and a grid of numbers, usually a 10 × 10, reporting, in the position `values[`*M*`][`*N*`]`, the profit or loss of this position if the event result was home: *M*, away: *N*.

`totals` **[ *object{string -> object}* ]:**

Information about the matched price and stake for each bet type.

`placers` **[ *list[string, ...]* ]:**

List of usernames of the MollyBet accounts that placed bets making this position.

`unknown_bets_num` **[ *int* ]:**

The number of bets on this position whose state is currently unknown.

`unknown_grid` **[ *object{*`ccy_code` *-> string,* `values` *-> list[list[float, ...], ...]}* ] (optional):**

An object with the same meaning of the `payoff_grid` reporting the possible profit/loss for bets which are in an unknown state, if any.

The objects returned in the `totals` object have the following attribute:

`bet_type_description` **[ *string* ]:**

An human-readable explanation of the bet type.

`got_price` **[ *float* ]:**

The weighted-average price obtained on this bet type.

`got_stake` **[ *list[string, float]* ]:**

The total stake placed on this bet type, as a [*currency code*, *amount*] 2-elements list.

`unknown_price` **[ *float* ] (optional):**

The weighted-average price for bets whose state is currently unknown.

`unknown_stake` **[ *list[string, float]* ] (optional):**

The total stake for bets whose state is currently unknown, as a [*currency code*, *amount*] 2-elements list.

**Example:**

After some bets have placed for *Borussia Dortmund* as home team winning, for a combined stake of 100 EUR at an average decimal price of 2.88, the following position grid could be returned:

```
{
  "payoff_grid": {
    "ccy_code": "EUR",
    "values": [
      [-100.00, -100.00, -100.00, /* ... other 7 values */ ],
      [188.00, -100.00, -100.00, /* ... other 7 values */ ],
      [188.00, 188.00, -100.00, /* ... other 7 values */ ],
      /* ... other 7 rows */
    ]
  },
  "totals": {
    "for,h": {
      "bet_type_description": "BV Borussia 09 Dortmund",
      "got_price": 2.88,
      "got_stake": ["EUR", 100.00],
      "unknown_price": null,
      "unknown_stake": ["EUR", 0.0],
    }
  },
  "placers": ["john", "jack"],
  "unknown_bets_num": 0,
  "unknown_grid": null
}
```

### *4.3.10 Return the order processing log*

**Method and URL:**

```
GET /v1/orders/<order_id>/log/
```

This API call returns a timeline of the operations performed by MollyBet to process an order, showing for instance why a certain account was preferred to others, what kind of problem were found, the timing of the information received.

**URL Parameters:**

`order_id`**:**

The ID of the order to get the log for.

**Request Parameters:**

`categories` **[ *list[string, ...]* ] (optional):**

The list of categories for which you wish to retrieve log messages.

Some categories might not be available to you. The list of available categories is visible in your preferences from the Customer Info endpoint.

Omitting this parameter is the same as requesting all available categories.

`page` **[ *int* ] (optional, default:** `1`**, minimum:** `1`**):**

Page number to retrieve

`page_size` **[ *int* ] (optional, default:** `100`**, minimum:** `1`**, maximum:** `1000`**):**

Maximum number of items per page

**Return Data:** A list of objects reporting the operation performed.

**Output Format:**

`timestamp` **[ *string* ]:**

The date and time the operation was performed.

`category` **[ *string* ] (optional):**

The message category. See list of order log categories. This will be absent for older log entries.

`message` **[ *string* ]:**

An human description of the operation performed.

**Example:**

```
[
  {
    "timestamp": "2016-04-27T08:51:16.828+00:00Z",
    "category": "account_selection",
    "message": "Requested Accounts: pin/AAAAA, bf/BBBBB, sing2/CCCCC"
  },
  {
    "timestamp": "2016-04-27T08:51:16.962+00:00Z",
    "category": "round_summary",
    "message": "Start of round 1: want £5.00 @ 1.3.
                Account sing2/CCCCC skipped: temporary problems
                Account bf/BBBBB: placing £5.00 @ 1.624
                IDs of bets being placed: 3695153.
                End of round 1."
  },
  {
    "timestamp": "2016-04-27T08:51:16.980+00:00Z",
    "category": "placement_summary",
    "message": "Placement summary: wanted £5.00 @ 1.3.
                Placement using bf/BBBBB: placing £5.00 @ 1.624
                IDs of bets being placed: 3695153.
                Placing: £5.00 @ 1.624."
  },
  {
    "timestamp": "2016-04-27T08:51:17.004+00:00Z",
    "category": "status_update",
    "message": "Bet 3695153 placing: £0.00 (of £4.96)
                Bet 3695153 done: £4.96 (of £4.96) @ 1.780
                Order closed: order_filled"
  }
]
```

## 4.4 Positions Requests

Informative API calls allowing to explore the payout grid of bet types.

4.4.1 Return information about a bet type, including win/loss grid 36

### *4.4.1 Return information about a bet type, including win/loss grid*

**Method and URL:**

```
GET /v1/sports/<sport>/bet_types/<bet_type>/
```

The win/loss grid is a grid, usually 10 × 10 elements, representing the result of a bet of the given type according to the result. The element `position[`*M*`][`*N*`]` represents the outcome of the bet if the result is home: *M*, away: *N*. Each element can be:

- `w`: the bet is winning;
- `l`: the bet is losing;
- `v`: the bet is refunded (e.g. a result 0-0 for a football bet of type `for,ah,h,0`);
- `v/w`: half of the stake is winning, half refunded (e.g. a result 0-0 for a football bet of type `for,ah,h,1`) ;
- `l/v`: half of the stake is losing, half refunded (e.g. a result 0-0 for a football bet of type `for,ah,h,-1`).

**URL Parameters:**

`sport`**:**

The code of the sport.

`bet_type`**:**

The code of the bet type.

**Request Parameters:**

`home_team` **[ *string* ] (optional):**

The name of the home team, to include in the human-readable bet type name, if useful.

`away_team` **[ *string* ] (optional):**

The name of the away team, to include in the human-readable bet type name, if useful.

**Output Format:**

`sport` **[ *string* ]:**

The human-readable name of the sport.

`bet_type_description` **[ *string* ]:**

The human-readable name of the bet type.

`position` **[ *list[list[string, ...], ...]* ]**

The win/loss code for each outcome of the bet.

**Example:**

The GET request:

```
/v1/sports/fb/bet_types/for,ah,h,3/?home_team=Arsenal&away_team=Liverpool
```

could return the following structure:

```
{
  "sport": "Football Full Time",
  "bet_type_description": "Arsenal +0.5/1 (Asian)",
  "winloss_grid": [
    ["w", "l/v", "l",   "l",   /* other 6 codes */ ],
    ["w", "w",   "l/v", "l",   /* other 6 codes */ ],
    ["w", "w",   "w",   "l/v", /* other 6 codes */ ],
    ["w", "w",   "w",   "w",   /* other 6 codes */ ],
    /* other 6 rows */
  ]
}
```

## 4.5 Customers Requests

These API calls allow creating and manipulating MollyBet accounts and to return information about either the currently logged-in account or about other MollyBet accounts (provided that the caller has the required permissions).

**4.5.1 Create a new MollyBet account** 37

**4.5.2 List all the MollyBet accounts known by the requesting user** 38

**4.5.3 Return information about a single MollyBet account** 38

**4.5.4 Change some of the attributes of a MollyBet account** 39

**4.5.5 Check if the current user can place bets with this IP** 39

**4.5.6 Return the list of the bookmaker accounts available to the customer** 39

**4.5.7 Get the accounting summary of a MollyBet account** 40

**4.5.8 Return the history of the credit limit changes of a MollyBet account** 40

**4.5.9 Request a refresh of the balance for a customer's bookie account** 41

**4.5.10 Get a provisioning key for two-step authentication** 41

**4.5.11 Check if a customer has enabled two-step authentication** 42

**4.5.12 Enable or disable two-step authentication for a customer** 42

### *4.5.1 Create a new MollyBet account*

**Method and URL:**

```
POST /v1/customers/
```

**Request Parameters:**

`password` **[ *string* ]:**

The password of the new user. Must be at least 8 characters long and contain both numbers and letters.

`group_id` **[ *int* ]:**

The numeric group ID of the new user. You can only create users in groups which you manage.

`name` **[ *string* ] (optional, default:** `""`**):**

The display name of the new user.

`credit_limit` **[ *list[string, float]* ] (optional, minimum:** `0`**, maximum:** `1000000000000000000`**):**

" The credit limit of the user, as a [*currency code*, *amount*] 2-items list.

`credit_limit_comment` **[ *string* ] (optional):**

Optional credit limit description.

`commission_rate` **[ *object{*`date_from` *-> string,* `rate_pc` *-> float}* ] (optional):**

The commission rate to apply to the user, with the date it starts being active.

`config` **[ *object* ] (optional, default:**
`{"login": true, "see_accounts": true, "see_bookies": true}`**):**

A map with configuration for this user.

`preferences` **[ *object* ] (optional):**

A map of user preferences.

`locked` **[ *bool* ] (optional):**

The user is locked due to multiple login failures.

`tags` **[ *object* ] (optional):**

An optional map with a set of tags to apply to the user.

Tags are key/value properties, encoded as a single object whose fields (the keys) are strings and whose values are also strings. Futher restrictions are:

1. Keys cannot be more than 64 characters long and cannot be empty;
2. Values cannot be more than 128 characters long (but can be empty);
3. There can be no more than 32 tags per customer.

`username` **[ *string* ]:**

The username of the new user. Must be unique and only contain alphanumeric characters.

`ccy_code` **[ *string* ]:**

The currency of the new user, as an ISO 4217 three-letter currency code.

**Return Data:** Data about the customer just created, in a format similar to the input parameters.

### *4.5.2 List all the MollyBet accounts known by the requesting user*

**Method and URL:**

```
GET /v1/customers/
```

**Request Parameters:**

`last_login` **[ *string* ] (optional, default:** `"true"`**):**

if "true", returns information about the last time each user logged in

`inactive` **[ *string* ] (optional, default:** `"true"`**):**

if "true" include inactive accounts

`permissions` **[ *string* ] (optional, default:** `"false"`**):**

if "true", return a set of permissions each user possesses

`customer_ids` **[ *string* ] (optional):**

comma-separated list of ids of customers to return

**Return Data:** A list of objects representing the data about the accounts known.

### *4.5.3 Return information about a single MollyBet account*

**Method and URL:**

```
GET /v1/customers/<username>/
```

**URL Parameters:**

`username`:

MollyBet account username to return.

**Return Data:** An object representing the data about the account.

### 4.5.4 *Change some of the attributes of a MollyBet account*

**Method and URL:**

```
PATCH /v1/customers/<username>/
```

**URL Parameters:**

`username`:

MollyBet account username to change.

**Request Parameters:**

The same accepted in Create Account.

**Return Data:** Data about the customer just updated, in a format similar to the input parameters.

### 4.5.5 *Check if the current user can place bets with this IP*

**Method and URL:**

```
GET /v1/customers/<username>/can_place_bets/
```

**URL Parameters:**

`username`:

MollyBet account username to check; only the current logged in user is supported.

**Return Data: An object with the following properties:**

- **can_place_bets (bool): a boolean indicating whether the**
  user can place bets from the IP address from which this request is made
- **reason (str): an optional hint about why the value of**
  can_place_bets was selected
- **ip_address (str): what mollybet API believes to be the**
  IP address of the request
- **country (str): what mollybet API believes to be the country**
  that corresponds to the IP address of the request

### 4.5.6 *Return the list of the bookmaker accounts available to the customer*

**Method and URL:**

```
GET /v1/customers/<username>/bookie_accounts/
```

**URL Parameters:**

`username`:

MollyBet account username to get info for.

**Return Data:** An objects as described in Group Accounts list.

### 4.5.7 *Get the accounting summary of a MollyBet account*

**Method and URL:**

```
GET /v1/customers/<username>/accounting_info/
```

The accounting summary includes the balances, profit/loss, and open stake of a MollyBet account.

The method returns a JSON list of objects with members:

- key: a short string describing the entry; provided entries are currently current_balance, open_stakes, today_pl, yesterday_pl, credit_limit, commission_rate;
- label: a human-readable label for the entry
- **unit: the unit of the entry: currency codes such as EUR, GBP for monetary**

  amounts, or % for percentages;
- value: the numeric amount of the entry

A sample output of the function could be the following JSON message:

```
[{"key": "current_balance",
  "label": "Current balance",
  "unit": "SGD",
  "value": 85024.7},
 {"key": "open_stakes",
  "label": "Open stakes",
  "unit": "SGD",
  "value": 3438.03},
 {"key": "credit_limit",
  "label": "Agent credit limit",
  "unit": "SGD",
  "value": 1000000.0},
 {"key": "commission_rate",
  "label": "Current commission rate",
  "unit": "%",
  "value": 0.25}]
```

**URL Parameters:**

`username`**:**

MollyBet account username to get info for.

### 4.5.8 *Return the history of the credit limit changes of a MollyBet account*

**Method and URL:**

```
GET /v1/customers/<username>/credit_limit_history/
```

**URL Parameters:**

`username`**:**

MollyBet account username to get info for.

**Return Data:** A list of objects with the previous changes to the credit limit

**Output Format:**

`amount` **[ *float* ]:**

the new credit limit; null if no limit is set;

`ccy_code` **[ *string* ]:**

the currency of the credit limit (usually the customer currency);

`timestamp` **[ *string* ]:**

the timestamp in which the credit limit was set;

`comment` **[ *text* ]:**

any comment set by the agent when the credit limit was set.

**Example:**

```
[{"amount": 15000.0,
  "ccy_code": "GBP",
  "timestamp": "2014-03-24T18:48:58.006352+00:00",
  "comment": "Credit limit set by agent"}]
```

### *4.5.9 Request a refresh of the balance for a customer's bookie account*

**Method and URL:**

```
POST /v1/customers/<username>/bookie_accounts/kick_balance/
```

**URL Parameters:**

`username`**:**

MollyBet account username to which the bookie account belongs

**Request Parameters:**

`account_id` **[ *int* ]:**

ID of the bookie account to refresh balance for.

**Return Data:** an object with the attribute `balance` containing the bookie account's newly-fetched balance.

### *4.5.10 Get a provisioning key for two-step authentication*

**Method and URL:**

```
GET /v1/customers/<username>/two_step_auth/provisioning_key/
```

This endpoint creates a key that can be used to set up two-step authentication for the customer. The elements of the key should be rendered into a QR code (see Key URI Format) that the customer can scan into their authenticator app. When generating the QR code, you may wish to define the `issuer` parameter as a customer-friendly string; this will be displayed within the app in place of the default `mollyapi` identifier.

This endpoint does not *enable* two-step authentication. You will need to make a subsequent call to update the two-step authentication status of the user.

**URL Parameters:**

`username`**:**

The unique username of the MollyBet customer.

**Output Format:**

`type` **[ *string* ]:**

The type of one-time password to be used for two-step authentication.

`label` **[ *string* ]:**

A unique identifier for this MollyBet account to be used by the authenticator app. The `mollyapi` prefix will hidden within the app so long as you define a friendly `issuer` when generating the QR code.

`secret` **[ *string* ]:**

The 16-character secret key that is shared between the MollyBet API and the user's authenticator app.

**Example:**

```
{
  "type": "totp",
  "label": "mollyapi:johndoe",
  "secret": "F2VOLX4REGLT5PXJ"
}
```

### 4.5.11 Check if a customer has enabled two-step authentication

**Method and URL:**

```
GET /v1/customers/<username>/two_step_auth/status/
```

**URL Parameters:**

`username`**:**

The unique username of the MollyBet customer.

**Output Format:**

`enabled` **[ *bool* ]:**

Whether the customer has enabled two-step authentication.

### 4.5.12 Enable or disable two-step authentication for a customer

**Method and URL:**

```
PATCH /v1/customers/<username>/two_step_auth/status/
```

When enabling two-step authentication, this endpoint will return a list of single-use recovery codes. These should be passed on to the customer for safe-keeping; they can be used in place of an authenticator token when the user does not have their authenticator device to hand.

**URL Parameters:**

`username`**:**

The unique username of the MollyBet customer.

**Request Parameters:**

`enabled` **[ *bool* ]:**

Whether two-step authentication should be enabled.

`token` **[ *string* ] (optional):**

A valid authenticator token. When disabling two-step authentication, a recovery code is also permitted. Agents can disable two-step authentication for their customers without providing a token.

`secret` **[ *string* ] (optional):**

The secret key that was added to the user's authenticator app. Only necessary when enabling two-step authentication.

**Example:**

After enabling two-step authentication:

```
["a7bcb-9f240", "ca066-996a7", "3a553-b5906"]
```

## 4.6 Groups Requests

4.6.1 Return the list of the known MollyBet account groups 43
4.6.2 Return information about a MollyBet group 43
4.6.3 Return the list of the bookmakers accounts available to a group 43
4.6.4 Set the new credit limit for a group 44
4.6.5 Return the history of the credit limit changes of a MollyBet group 45

### *4.6.1 Return the list of the known MollyBet account groups*

**Method and URL:**

```
GET /v1/groups/
```

Every MollyBet account belongs to an account group. Normal accounts can see the group they belong to, certain accounts can be used to manage more than one group.

**Return Data:** A list of objects with information about each group.

**Output Format:**

`id` **[ *integer* ]:**

Numeric identifier of the group.

`name` **[ *string* ]:**

Human-readable name of the group.

`balance` **[ *list[string, float]* ]:**

The balance of the group (sum of the balances of the accounts in the group) as a [ *currency code*, *amount* ] 2-items list.

`credit_limit` **[ *list[string, float]* ]:**

The credit limit of the group, if set, as a [ *currency code*, *amount* ] 2-items list.

### *4.6.2 Return information about a MollyBet group*

**Method and URL:**

```
GET /v1/groups/<group_id>/
```

Every MollyBet account belongs to an account group. Normal accounts can see the group they belong to, certain accounts can be used to manage more than one group.

**URL Parameters:**

`group_id`**:**

The id of the MollyBet group to return info about.

**Return Data:** An object with information about the group as in Group List output.

### *4.6.3 Return the list of the bookmakers accounts available to a group*

**Method and URL:**

```
GET /v1/groups/<group_id>/bookie_accounts/
```

**URL Parameters:**

`group_id`**:**

The id of the group to get the accounts list for.

**Return Data:** A list of objects reporting the accounts available.

**Output Format:**

`id` **[ *integer* ]:**

The numeric id of the account.

`bookie` **[ *string* ]:**

The bookmaker code of the account.

`username` **[ *string* ]:**

The account username of the bookie account.

`enabled` **[ *bool* ]:**

`true` if the account is enabled for placement.

`stake_share` **[ *float* ]:**

The percentage of stake that the account will not place (MollyBet will adjust the requested stake accordingly).

`agent` **[ *string* ]**

The name of the agent providing the account.

`max_bet` **[ *list[string, float]* ]:**

The maximum bet accepted on this account, if the information is available.

`price_scheme` **[ *string* ]:**

The type of price accepted by the account.

`ccy_code` **[ *string* ]:**

The currency accepted by the account for placement.

`balance` **[ *list[string, float]* ]**

The last account balance known by MollyBet.

### *4.6.4 Set the new credit limit for a group*

**Method and URL:**

```
PATCH /v1/groups/<group_id>/
```

The new limit takes immediately effect.

**URL Parameters:**

`group_id`**:**

The id of the MollyBet group to modify.

**Request Parameters:**

`credit_limit` **[ *list[string, float]* ] (optional):**

The credit limit of the group.

`credit_limit_comment` **[ *string* ] (optional):**

Optional credit limit description.

### *4.6.5 Return the history of the credit limit changes of a MollyBet group*

**Method and URL:**

```
GET /v1/groups/<group_id>/credit_limit_history/
```

**URL Parameters:**

`group_id`**:**

The id of the MollyBet group to get information for.

**Return Data:** A list of structures as in Customer Credit Limit History.

## 4.7 Transfers Requests

**4.7.1 Register a new transfer between the agent and a customer 45**

**4.7.2 Return the history of transfers performed by an agent 45**

### *4.7.1 Register a new transfer between the agent and a customer*

**Method and URL:**

```
POST /v1/transfers/
```

**Request Parameters:**

`username` **[ *string* ]:**

The customer the transfer is being register for.

`amount` **[ *float* ]:**

The amount transferred, in the customer account currency.

Positive values represent money given by the agent to the customer; negative values represent money received by the agent from the customer.

`discount` **[ *float* ] (optional, default:** `0.0`**):**

An adjustment for the new balance of the customer after the transfer, on top of the `amount`.

Positive values are always in the customer's favour. You send `amount`, they receive `amount + discount`.

`post_date` **[ *ISO 8601 date string* ] (optional):**

The date the transfer takes effect. Optional. Deprecated

`description` **[ *string* ] (optional, default:** `""`**):**

An optional memo for the transfer.

`only_if_available` **[ *bool* ] (optional, default:** `false`**):**

Disallow the transfer if it would leave the customer with a negative balance.

### *4.7.2 Return the history of transfers performed by an agent*

**Method and URL:**

```
GET /v1/transfers/
```

**Request Parameters:**

`username` **[ *string* ] (optional):**

The user to get transfer from.

Must be a customer of the agent. If omitted, retrieves transfers for all users.

`date_from` **[ *ISO 8601 date string* ] (optional):**

Return transfers from this date on.

`date_to` **[ *ISO 8601 date string* ] (optional):**

Return transfers until this date.

**Return Data:** A list of objects with the data of the requested transfers.

**Output Format:**

`from_username` **[ *string* ]**

The MollyBet account username of the agent that performed the transfer.

`to_username` **[ *string* ]**

The MollyBet account username of the customer that received the transfer.

`to_group` **[ *string* ]**

The MollyBet account group of the customer that received the transfer.

`post_date` **[ *ISO 8601 date string* ]**

The date the transfer took effect.

`ccy_code` **[ *string* ]**

The ISO 4217 3-letters code of the currency of the transfer.

`description` **[ *string* ]**

The description provided when the transfer was created.

`amount_from` **[ *float* ]**

How much the agent sent. This is the same as the `amount` set when registering the transfer.

`amount_to` **[ *float* ]**

The negative of how much the customer received (i.e. how much was deducted from the customer's balance).

`discount` **[ *float* ]**

The discount applied to the transfer (how much more the customer received than the agent sent).

For every transfer, `amount_to` + `amount_from` + `discount` = 0.

## 4.8 Miscellaneous Requests

4.8.1 Return the list of exchange rates between GBP and other currencies 46

### 4.8.1 *Return the list of exchange rates between GBP and other currencies*

**Method and URL:**

```
GET /v1/xrates/
```

**Return Data:** A list of objects with information about each exchange rate.

**Output Format:**

`ccy` **[ *string* ]:**

The ISO 4217 code for the currency.

`rate` **[ *number* ]:**

The exchange rate between GBP and the currency, i.e.

*amount in ccy = amount in GBP × rate*

**Example:**

```
[{"ccy": "USD", "rate": 1.271},
 {"ccy": "GBP", "rate": 1.0},
 {"ccy": "EUR", "rate": 1.1197}]
```

# 5 Asynchronous Client Message Types

MollyBet API users who have successfully made an Asynchronous connection can communicate by sending client messages to the MollyBet API.

**5.1 Messages** **48**

5.1.1 `register_event`: register to receive updates for an event 48

5.1.2 `unregister_event`: stop receiving updates for an event 48

5.1.3 `list_registered_events`: list all registered events 48

5.1.4 `echo`: test connection by echoing back data 49

## 5.1 Messages

### 5.1.1 `register_event`: *register to receive updates for an event*

Register to an event with primary key `(sport, event_id)`.

A client will typically register to one or more events, as described in General Workflow, and start receiving asynchronous update messages with respect to those events.

**Fields:**

`sport` **[ *string* ]**

The sport of the event.

`event_id` **[ *string* ]**

The opaque ID of the event.

**Example:**

```
["register_event", "fb", "2016-06-17,417,6228"]
```

### 5.1.2 `unregister_event`: *stop receiving updates for an event*

Unregisters from an event with primary key `(sport, event_id)`.

When a client unregisters, it stops receiving further update messages with respect that event.

**Fields:**

`sport` **[ *string* ]**

The sport of the event.

`event_id` **[ *string* ]**

The opaque ID of the event.

**Example:**

```
["unregister_event", "fb", "2016-06-17,417,6228"]
```

### 5.1.3 `list_registered_events`: *list all registered events*

List all the events a client is registered to.

**Example:**

```
["list_registered_events"]
```

### *5.1.4* `echo`*: test connection by echoing back data*

The `echo` command is used to allow the client to send ping/pong requests to the server, to test the connection. Normally, websocket client libraries should provide access to the base WebSocket protocol PING and PONG messages, but in case the client library has no such API (e.g. web browsers), this command provides a useful alternative.

**Fields:**

Any fields provided, in addition to the command name, itself will be included in the command response.

**Example:**

```
> ["echo", 123, "abc"]
< {"ts":1526909238.0,"data":[["response",{"status":"ok","data":[123,"abc"]}]]}
```

# 6 Asynchronous Server Message Types

As explained in Asynchronous Server Message Format, the asynchronous messages sent by MollyBet API to your software have a structure:

```
{
    "ts": timestamp,
    "data": [
        [ code1, payload1 ],
        [ code2, payload2 ],
        ...
    ]
}
```

This section explains, for each `code` type, the format of the received `payload`.

**6.1 Connection Messages** **51**

6.1.1 `sync`: client initialised 51

6.1.2 `response`: websocket command completed 51

6.1.3 `info`: stream connection information 52

6.1.4 `connection_closed`: client is being disconnected 52

**6.2 Event Messages** **52**

6.2.1 `event`: event added or updated 53

6.2.2 `remove_event`: event no longer exists 53

6.2.3 `event_time`: current time of an in-running event 54

6.2.4 `event_score`: current score of an in-running event 55

6.2.5 `event_red_cards`: current red cards of an in-running event 55

6.2.6 `offer`: offer added or updated 55

6.2.7 `remove_offer`: offer no longer exists 57

**6.3 Betslips Messages** **57**

6.3.1 `betslip`: betslip opened or updated 57

6.3.2 `pmm`: price and min/max stake returned by a bookie 58

6.3.3 `betslip_closed`: betslip no longer exists 59

**6.4 Orders Messages** **59**

6.4.1 `order`: an order has been placed or updated 59

6.4.2 `bet`: bet placed or updated 59

**6.5 Other Messages** **60**

6.5.1 `balance`: customer balance and open stake 60

6.5.2 `xrate`: exchange rate to GBP updated 60

## 6.1 Connection Messages

6.1.1 `sync`: client initialised 51
6.1.2 `response`: websocket command completed 51
6.1.3 `info`: stream connection information 52
6.1.4 `connection_closed`: client is being disconnected 52

### *6.1.1* `sync`*: client initialised*

Reception of this message delineates the end of the initial state transfer from the MollyBet servers to the client.

**Fields:**

`token` **[ *string* ]**

The token of the current session.

**Example:**

```
{
    "token": "g8dsmnslx17pvsb3ufgy5zqyapwjvpdw"
}
```

### *6.1.2* `response`*: websocket command completed*

The outcome of the last command sent to the websocket. The content of the message depends on the command sent: see Asynchronous Client Message Types

Note that there is a strong guarantee that this message will be received after the end of the requested operation. For instance, registering to an event will first transfer all the pre-existing offers to your client before sending the corresponding response message.

**Fields:**

`status` **[ *string* ]**

The status of the command: `ok` or `error`.

`code` **[ *string* ]**

A code for the error; only sent in case the last command failed.

`data` **[ *any* ]**

Response data, whose type and content are specific to each request.

**Example:**

```
{
    "status": "ok",
    "data": {
        "registered_events": [
            ["tennis", "2016-11-02,64630,64456"],
            ["fb", "2016-11-02,27702,27716"]
        ]
}

{
    "status": "error",
    "code": "invalid_event"
}
```

### 6.1.3 `info`: *stream connection information*

Sent periodically with miscellaneous information about the current websocket connection.

**Fields:**

`queue_size` **[ *int* ]**

This indicates how slow the client is to read data from the server. Lower numbers are better. If this number grows too large you may be disconnected.

`registered_events` **[ *int* ]**

The total number of events currently registered for updates over this particular connection. The complete list of events may be requested using the list_registered_events call. This can be useful for managing your group register_event limits.

`max_queue_size` **[ *int* ]**

The maximum size (capacity) of the server internal queue for async messages still waiting to be transmitted. This is a constant. If the number of queued messages exceeds this values, the server closes the connection, on the theory that the client is a slow reader and with so many messages in the queue it means that messages must be arriving to client with so much delay that they are no longer useful.

`queue_size_max` **[ *int* ]**

This reports the maximum value, over the interval between this `info` message and the previous one, of the queue (used) size. If this value ever becomes in the same order of magnitude as `max_queue_size` then it is a sign of a problem. Either the client is too slow to read, or the network connection from client to Mollybet API is poor.

**Example:**

```
{
    "queue_size": 5,
    "registered_events": 87,
    "max_queue_size": 32768,
    "queue_size_max": 1,
}
```

### 6.1.4 `connection_closed`: *client is being disconnected*

The connection is being closed by the server. It has no content.

## 6.2 Event Messages

**6.2.1** `event`: **event added or updated** **53**

**6.2.2** `remove_event`: **event no longer exists** **53**

**6.2.3** `event_time`: **current time of an in-running event** **54**

**6.2.4** `event_score`: **current score of an in-running event** **55**

**6.2.5** `event_red_cards`: **current red cards of an in-running event** **55**

**6.2.6** `offer`: **offer added or updated** **55**

**6.2.7** `remove_offer`: **offer no longer exists** **57**

These are messages about the events MollyBet currently knows about. By default, MollyBet API sends a list of all available events upon asynchronous channel connection, and maintains that list as long as the connection is active.

Additional information about an event is sent as soon as register_event is called, up until the point where unregister_event is called. This information includes the prices available from your bookies, and for in-running events also scores, red cards and event times.

### 6.2.1 `event`: *event added or updated*

Asserts that an event with primary key `(sport, event_id)` exists, and has the details provided.

A client will typically store this information in a hash table with each key being a `(sport, event_id)` pair. If an event is received and its key already exists in the table, then it must replace the pre-existing entry. If a remove_event message is received, then the key must be removed from the table.

**Fields:**

`sport` **[ *string* ]**
  The sport of the event.

`event_id` **[ *string* ]**
  The opaque id of the event.

`home` **[ *string* ]**
  The name of the home team.

`away` **[ *string* ]**
  The name of the away team.

`competition_id` **[ *string* ]**
  The ID of the competition of which this event is part.

`competition_name` **[ *string* ]**
  The name of the competition of which this event is part.

`competition_country` **[ *string* ]**
  The country associated with the competition of which this event is part, as a two-letter ISO 3166 country code.

`ir_status` **[ *string* ]**
  The in-running status of the event. This will be either `pre_event` or `in_running`.

`start_time` **[ *string* ]**
  The time at which we believe the event will start, or `null` if it is unknown.

**Example:**

```
{
    "sport": "fb",
    "event_id": "2016-06-17,417,6228",
    "home": "Example Home Team 1",
    "away": "Example Home Team 2",
    "competition_id": "1851",
    "competition_name": "Test Division 3",
    "competition_country": "CY",
    "ir_status": "pre_event",
    "start_time": "2016-06-17T16:30:00+00:00"
}
```

### 6.2.2 `remove_event`: *event no longer exists*

An event with primary key `(sport, event_id)` which has previously existed in our system has ceased to exist.

This may be because the event has finished, because we no longer have offers for that event, or because the event has been merged with another.

**Fields:**

`sport` **[ *string* ]**
: The sport of the event.

`event_id` **[ *string* ]**
: The opaque ID of the event.

`home` **[ *string* ]**
: The name of the home team.

`away` **[ *string* ]**
: The name of the away team.

**Example:**

```
{
    "sport": "fb",
    "event_id": "2016-06-17,417,6228",
    "home": "Example Home Team 1",
    "away": "Example Home Team 2",
}
```

### 6.2.3 `event_time`*: current time of an in-running event*

Notification of our best guess of the current event time for in-running events in which you have registered an interest.

Note that for events with multiple sub-sports (e.g., `fb`, `fb_ht`, `fb_corn` etc) we will only send time information for the main event (in this instance, `fb`).

Because our guess at the event time is aggregated from a number of sources, it may behave in unexpected ways, such as going backwards when we receive more information that adjusts our idea of the event time.

**Fields:**

`sport` **[ *string* ]**
: The sport of the event.

`event_id` **[ *string* ]**
: The opaque id of the event.

`time` **[ *list[string, int]* ]**
: For football, this is either of:

- a 2-element list of the form [*half*, *mins*], where *mins* is the number of elapsed minutes and *half* is `1h`, or `2h`, representing the first half or the second half respectively. During half time, the time will be `ht`.
- `null` if the time is unknown.

**Example:**

```
{
    "sport": "fb",
    "event_id": "2016-06-18,2879,8302",
    "time": ["1h", 35]
}
```

### 6.2.4 `event_score`: *current score of an in-running event*

Notification of our best guess of the current event score for in-running events in which you have registered an interest. As with `event_time`, the current score may sometimes behave in unexpected ways.

**Fields:**

`sport` **[ *string* ]**

The sport of the event.

`event_id` **[ *string* ]**

The opaque id of the event.

`score` **[ *list[int, int]* ]**

For football, this is either of:

- a 2-element list of the form [*home*, *away*], where *home* and *away* represent the number of goals scored by the home and away teams (or number of corners awarded, depending on the sport).
- `null` if the score is unknown.

**Example:**

```
{
    "sport": "fb",
    "event_id": "2016-06-18,2879,8302",
    "score": [1, 1]
}
```

### 6.2.5 `event_red_cards`: *current red cards of an in-running event*

Notification of our best guess of the current number of red cards received by each team in a football match. As with `event_time` and `event_score`, this may behave in unexpected ways.

**Fields:**

`sport` **[ *string* ]**

The sport of the event.

`event_id` **[ *string* ]**

The opaque id of the event.

`score` **[ *list[int, int]* ]**

For football, this is either of:

- a 2-element list of the form [*home*, *away*], where *home* and *away* represent the number of red cards received by the home and away teams.
- `null` if the red cards are unknown.

**Example:**

```
{
    "sport": "fb",
    "event_id": "2016-06-18,2879,8302",
    "score": [1, 1]
}
```

### 6.2.6 `offer`: *offer added or updated*

Asserts that an offer with primary key `(bookie, market, selection)` exists, and has the details provided.

Your local offers state will typically be stored in a hash table with each key being a `(bookie, market, selection)` triplet. If a offer is received and its triplet already exists in the table, then it must replace the pre-existing entry. If a remove_offer message is received, then the key must be removed from the table.

Note that a the bet type associated with a specific triplet may change over time, for example when a bookie updates the score of an in-running game, or changes the handicap line.

**Fields:**

`bookie` **[ *string* ]**

The bookie making the offer.

`market` **[ *string* ]**

Identifier used to disambiguate bets. Occasionally, bookies will offer the same bet multiple times. In those cases, they will have different market/selection IDs. This information is needed when the offer is removed, so that the client can tell which of the two (or more) offers has been removed.

`selection` **[ *string* ]**

Same as `market`.

`sport` **[ *string* ]**

The sport of the event.

`event_id` **[ *string* ]**

The opaque id of the event.

`bet_type` **[ *string* ]**

The bet type of the offer.

`in_running` **[ *boolean* ]**

`true` if this is an in-running offer.

`price_list` **[ *list[object, ...]* ]**

List of prices available from the bookie. Some bookies (for example, exchanges) will often have multiple prices available. Each price is an object with a single key called `bookie` (unlike PMMs, offers will not have an effective price). `bookie` is an object with three keys:

`price` **[*float*]**

Available price in decimal odds. See Price and Stake.

`min` **[*string*, *float*]**

Minimum stake. May be `null` if unknown. See Price and Stake.

`max` **[*string*, *float*]**

Maximum stake. May be `null` if unknown. See Price and Stake.

Exchanges may also exceptionally have empty price lists, which indicate that only an exchange order is available.

**Example:**

```
{
    "bookie": "bet123",
    "market": "market-id",
    "selection": "selection-id",
    "sport": "fb",
    "event_id": "2016-06-18,2879,8302",
    "bet_type": "for,h",
    "in_running": false
    "price_list": [
        {
            "bookie": {
                "price": 1.8,
                "min": 40
                "max": 40000,
            }
        }
    ]
}
```

### 6.2.7 `remove_offer`: *offer no longer exists*

An offer with primary key `(bookie, market, selection)` which has previously existed in our system has ceased to exist.

**Fields:**

`bookie` **[ *string* ]**
  Part of the primary key of the removed offer.

`market` **[ *string* ]**
  Part of the primary key of the removed offer.

`selection` **[ *string* ]**
  Part of the primary key of the removed offer.

`sport` **[ *string* ]**
  The sport of the event.

`event_id` **[ *string* ]**
  The opaque id of the event.

`bet_type` **[ *string* ]**
  The bet type for the offer.

**Example:**

```
{
    "bookie": "bet123",
    "market": "market-id",
    "selection": "selection-id",
    "sport": "fb",
    "event_id": "2016-06-18,2879,8302",
    "bet_type": "for,h",
}
```

## 6.3 Betslips Messages

These messages update the state of an open betslip. They are sent in response to betslips requests.

**6.3.1** `betslip`: **betslip opened or updated** **57**

**6.3.2** `pmm`: **price and min/max stake returned by a bookie** **58**

**6.3.3** `betslip_closed`: **betslip no longer exists** **59**

### 6.3.1 `betslip`: *betslip opened or updated*

A betslip has been opened by an Open Betslip command.

The format of this message is the same returned in the synchronous response.

### 6.3.2 `pmm`*: price and min/max stake returned by a bookie*

A bookie has returned a PMM for an offer on a betslip.

**Fields:**

`betslip_id` **[ *string* ]**

The random UUID of the betslip that requested the PMM.

`sport` **[ *string* ]**

The sport of the event.

`event_id` **[ *string* ]**

The opaque id of the event.

`bookie` **[ *string* ]**

The bookie of the bookie account.

`username` **[ *string* ]**

The username of the bookie account.

`bet_type` **[ *string* ]**

The bet type of the PMM.

`status` **[ *object* ]**

An object describing the result of the PMM. This will contain at least one key, `code`. `code` will be one of:

`success`

The bookie account was successfully queried and the attached price list has the latest prices from the bookie.

`error` **or** `failed`

We were unable to either reach the bookie due to technical reasons (e.g. network issues, or problems with either our servers or the bookie's), or that the response we received from the bookie was rejected (e.g. they returned the wrong data).

For either of these statuses, the parent object will always contain a second key, `reason`, with more details about the error.

Note that an error from the bookie may result in a temporary blacklist on a bookie account or offer. In such cases MollyBet will replace the account on the betslip with another one and try again, if possible.

`blacklist`

This account is currently blacklisted and we're searching for a bookie account that can replace it.

`price_list` **[ *list[object{*`bookie`*: object,* `effective`*: object}, ...]* ]**

List of available prices offered on this account. `bookie` is the prices list as offered by the bookie, `effective` takes modifiers such as commissions or referral fees into account. Both lists are PMM objects with attributes:

`price` **[ *float* ]:**

The decimal price of the offer.

`min` **[ *list[string, float]* ]:**

The minimum statke that can be placed, as a 2-items [ *currency code*, *amount* ] list.

`max` **[ *list[string, float]* ]:**

The maximum statke that can be placed, as a 2-items [ *currency code*, *amount* ] list.

**Example:**

```
{
  "betslip_id": "b2e936565fea4d778c38269ab1667f17",
  "sport": "fb",
  "event_id": "2016-10-26,245,234",
  "bookie": "bf",
  "username": "_40a78ed8_",
  "bet_type": "for,h",
  "status": {"code": "success"},
  "price_list": [
    {
      "bookie": {"price": 2.16, "max": ["GBP", 5.4], "min": ["GBP", 2.0]},
      "effective": {"price": 2.1368, "max": ["GBP", 5.4], "min": ["GBP", 2.0]}
    },
    {
      "bookie": {"price": 2.12, "min": ["GBP", 2.0], "max": ["GBP", 278.57]},
      "effective": {"price": 2.0976, "min": ["GBP", 2.0], "max": ["GBP", 278.57]}
    }
  ]
}
```

### 6.3.3 `betslip_closed`*: betslip no longer exists*

A betslip was closed.

**Fields:**

`betslip_id` **[ *string* ]**

The id of the betslip closed.

`close_reason` **[ *string* ]**

The reason the betslip was closed (such as explicit close request or timeout).

## 6.4 Orders Messages

6.4.1 `order`: **an order has been placed or updated** 59

6.4.2 `bet`: **bet placed or updated** 59

### 6.4.1 `order`*: an order has been placed or updated*

This is sent when a new order has been placed, either by the current customer, or by another customer in the current MollyBet group (only if `spy_on_group` is enabled). It is also sent whenever any of the fields of the order, such as *status*, changes value.

The message consists of an order structure in the same format as order-structure, except that the *bets* field is omitted (since a separate *bet* async message is provided, see async_bet).

### 6.4.2 `bet`*: bet placed or updated*

Either a new bet has been placed, for the specified order, or the data of a bet already existing in the order is being replaced with new data.

Properties of a bet that may be updated include the `status` (for example, if a pending bet fails to land), `got_stake` (if the bookie gives us less stake than we originally requested), `profit_loss` when the bet becomes settled or reconciled, etc.

The fields of this message are the same as in bet-structure.

## 6.5 Other Messages

6.5.1 `balance`: customer balance and open stake 60

6.5.2 `xrate`: exchange rate to GBP updated 60

### 6.5.1 `balance`: customer balance and open stake

The current MollyBet balance and total stake from unsettled bets.

**Fields:**

`balance` **[ *list[string, float]* ]**

The MollyBet balance of the current user, as a 2-element list, where the first element is a currency code and the second element an amount in that currency.

`open_stake` **[ *list[string, float]* ]**

The open stake of the current user, as a 2-element list, where the first element is a currency code and the second element an amount in that currency.

**Example:**

```
{
    "balance": ["EUR", 10000.1],
    "open_stake": ["EUR", 152.55]
}
```

### 6.5.2 `xrate`: exchange rate to GBP updated

**Fields:**

`ccy` **[ *string* ]**

The ISO 4217 code for the currency.

`rate` **[ *number* ]**

The exchange rate between GBP and the currency.

**Example:**

```
{
    "ccy": "EUR",
    "rate": 1.27
}
```

# 7 Appendix

## 7.1 List of Sports

Whenever a reference is made to a sport, it is one of:

`fb`
: Football, over 90 minutes.

`fb_ht`
: Football, first half only.

`fb_et`
: Football, extra time only.

`fb_corn`
: Football, the number of corners awarded in 90 minutes.

`fb_corn_ht`
: Football, the number of corners, first half only.

`tennis`
: Tennis.

`basket`
: Basketball.

`af`
: American football (includes Canadian football).

`ru`
: Rugby Union.

`rl`
: Rugby League.

`ih`
: Ice Hockey.

## 7.2 List of Bet Types

Bet types are strings consisting of a comma separated list of terms describing the bet.

The first term is `for` or `against`, depending on whether it's a back or a lay bet respectively. Handicaps always refer to the home team.

Asian handicaps are multiplied by 4 so they're always integers.

Available bet types are:

`for,h` **(**`for,a`**)**
: The home team (away team) will win.

`for,d`
: The game will be a draw.

`for,win_90,h` **(**`for,win_90,a`**)**
: The home team (away team) will win by 90 minutes, without going into extra time.

`for,dnb,h` **(**`for,dnb,a`**)**
: The home team (away team) will win, bet is void in the event of a draw.

`for,ml,h` **(**`for,ml,a`**)**

The home team (away team) will win, bet is void in the event of a draw. `ml` is short for *moneyline*. This differs from `dnb` in that we expect a draw can't happen and is simply an indication of that to customers.

`for,overeq,`***n*** **(**`for,undereq,`***n*****)**

The game will have *n* or more (less) goals in total. *n* must be an integer.

`for,over,`***n*** **(**`for,under,`***n*****)**

The game will have more (less) than *n* goals. *n* must be non-integral (*e.g.*, `2.5`).

`for,ah,h,`***n*** **(**`for,ah,a,`***n*****)**

The home team (away team) will win, once an asian handicap of code *n* has been applied to the home team.

`for,ahover,`***n*** **(**`for,ahunder,`***n*****)**

The total goals will be more (less) than handicap code *n*.

`for,tahover,h,`***n*** **(**`for,tahover,a,`***n*****)**

Home team (away team) will score more than handicap code *n* goals.

`for,tahunder,h,`***n*** **(**`for,tahunder,a,`***n*****)**

Home team (away team) will score less than handicap code *n* goals.

`for,eh,h,`***n*** **(**`for,eh,a,`***n*****)**

The home team (away team) will win with an "english handicap" of *n*. That's to say, the bet wins if and only if the final score *x*:*y* is such that $x+n > y$ ($x+n < y$).

`for,eh,d,`***n***

The match will draw with an "english handicap" of *n*. That's to say, the bet wins if and only if the final score *x*:*y* is such that $x+n = y$.

`for,cs,`***x***`,`***y***

The score will be *x* to the home team, and *y* to the away team where *x* and *y* are both integers.

`for,othercs,`***x***`,`***y***

Either the home team will score strictly more than *x* goals, or the away team will score strictly more than *y* goals. ("more than" but **not** "equal to").

`for,othercs,`***w***`,`***x***`,`***y***`,`***z***

Either the home team will score less than *w* or more than *y* goals or the away team will score less than *x* goals or more than *z* goals.

`for,odd` **(**`for,even`**)**

The total number of goals scored will be an odd number (even number).

`for,odd,h` **(**`for,even,h`**)**

The total number of goals scored by the home team will be an odd number (even number).

`for,odd,a` **(**`for,even,a`**)**

The total number of goals scored by the away team will be an odd number (even number).

`for,gr,`***x***`,`***y***

The total number of goals shall lie between *x* and *y* inclusive. It is permissible to emit a value `inf` for *y* representing infinity.

`for,teamgr,h,`***x***`,`***y*** **(**`for,teamgr,a,`***x***`,`***y*****)**

goal range between *x* and *y* inclusive for home team (away team). Can use `inf` for infinity.

`for,wg,h,`***n*** **(**`for,wg,a,`***n*****)**

The home side (away side) will win, and will score at least *n* goals in doing so.

`for,wintonil,h` **(**`for,wintonil,a`**)**

Home team (away team) to win and keep a clean sheet.

`for,wintonil,h,no` **(**`for,wintonil,a,no`**)**

Either the home team (away team) will score, or neither team will score.

`for,wm,h,`***x***`,`***y*** **(**`for,wm,a,`***x***`,`***y*****)**

The home team (away team) will win by a margin of between *x* and *y* goals inclusive. Use `inf` for infinity.

`for,fg,no_goal`

In first goal markets, there will be no goal (equivalent to `for,cs,0,0`).

`for,swm,no_goal`

In score and winning margin markets, there will be no goal (equivalent to `for,cs,0,0`).

`for,swm,sd`

In score and winning margin markets, there will be a score draw (i.e., any draw excluding 0-0).

`for,uswin,h` **(**`for,uswin,a`**)**

Offered by some American bookies. The home team (away team) will win. In the event of a draw, it is treated as two bets, each of half the chosen stake, one of which wins, one of which loses.

`for,quatro,h,o,`***x*****, (**`for,quatro,a,o,`***x*****)**

The home team (away team) will win, and over *x* goals will be scored in the match (*x* should not be an integer).

`for,quatro,h,u,`***x*** **(**`for,quatro,a,u,`***x*****)**

The home team (away team) will win, and under *x* goals will be scored in the match (*x* should not be an integer).

`for,quatro,hd,o,`***x*** **(**`for,quatro,ad,o,`***x*****)**

The home team (away team) will win or draw, and over *x* goals will be scored in the match (*x* should not be an integer).

`for,quatro,hd,u,`***x*** **(**`for,quatro,ad,u,`***x*****)**

The home team (away team) will win or draw, and under *x* goals will be scored in the match (*x* should not be an integer).

`for,dc,h,d` **(**`for,dc,a,d`**)**

Either the home team (away team) will win, or the match will be drawn (effectively, laying the opposite team win).

`for,dc,h,a`

Either the home team will win, or the away team will win (effectively, laying the draw).

`for,clean,h,(yes|no)` **(**`for,clean,a,(yes|no)`**)**

The home team (away team) will/won't keep a clean sheet.

`for,clean,h` **(**`for,clean,a`**)**

The home team (away team) will keep a clean sheet.

`for,clean,both`

Both teams will keep a clean sheet (same as `for,cs,0,0`).

`for,clean,either`

One or both of the teams will keep a clean sheet.

`for,clean,neither`

Neither team will keep a clean sheet.

`for,clean,one`

One (but not both) of the teams will keep a clean sheet.

`for,score,h,(yes|no)` **(**`for,score,a,(yes|no)`**)**

The home team (away team) will score or not.

`for,score,both`

Both teams will score.

`for,score,both,no`

Both teams will not score, i.e., either the score is 0-0 or only one team will score.

`for,score,either`

One or both teams will score.

`for,score,neither`

Neither team will score (same as `for,cs,0,0`).

`for,score,one`

One (but not both) of the teams will score.

`for,awm,`***n***

Absolute win margin. Wins if *n* = abs(*home_goals* - *away_goals*), *e.g.*, 1-0 or 0-1 are both `for,awm,1`.

There is some overlap between bet types, and some bookies may offer multiple bet types that equate to the same bet. The bet type we display will usually be the one that's closest to the text on the bookie's web site, even if there are other bet types we could have chosen.

A bet is settled when we (and the bookie) know the result of that bet, and thus how much the bet won or lost. For in-running bets, Asian bookies generally allow you to bet on the outcome of the remainder of the match, discarding any goals scored up until this point. It is therefore important to know the current score, and this is reflected in the bet type. For bets settled in this manner, we use the bet type:

```
for,ir,x,y,type
```

where *x* and *y* are the number of goals scored by the home and away teams respectively, and *type* is taken from the above list. Thus some example bet types for an in-running match where the score is currently 3-1 are:

```
for,ir,3,1,ah,h,0
for,ir,3,1,ahunder,19
for,a
for,cs,4,1
```

Note that the last two don't have the `ir,`*x*,*y* part. That is because they are bet types that settle on the result of the full 90 minutes, not on the result of the remainder of the match. Note also that the second of those is an over/under handicap bet, which Asian bookies generally settle on the whole match, not the remainder of the match. However, for historical reasons, our bet type for that includes the score anyway.

`for,X,Y`

Where X and Y are h (home team wins), d (draw), or a (away team wins). A bet on the combined result at half-time (X), and full-time (Y).

For example, the bet of type `for,h,d` wins if the home team is winning at the end of the first half AND the game ends in draw, at full time.

### *7.2.1 Tennis bet types*

Tennis bet types are of the form:

```
for,time_type,time_value,void_rule,rest_of_bet_type
```

The *time_type* and *time_value* fields indicate the time period for which this bet is relevant. *time_type* is currently always `tset` incidcating that the *time_value* refers to a number of sets, although in future this may also be `tgame` or `tpoint` to refer to individual games or points, respectively. *time_value* is either an integer referring to (currently) the set number, or the string `all`, referring to the entire game.

The *void_rule* is an indicator of the circumstances under which the bookie will void the bet. This is usually caused by a player pulling out due to injury. The currently allowed values are:

`vwhole`
: The bet will void unless the entire match is played to conclusion.

`vset1`, `vset2`, `vset3`, **etc.**
: The bet will void unless the relevant set has been completed.

`vgame1`, `vgame2`, `vgame3`, **etc.**
: The bet will void unless the relevant game has been completed.

`vpoint1`
: The bet will void unless at least one point has been won.

`vwhatever`
: This is a special kind of void rule that can be used only when opening a betslip (and placing orders from that betslip). It means we want the betslip to accept offers for any void rule.

Example tennis bet types:

`for,tset,all,vset1,p1`
: Player 1 will win the match.

`for,tset,1,vset1,p1`
: Player 1 will win the first set.

`for,tset,1,vwhole,p1`
: Player 1 will win the first set, but the bookie will void the bet if one of the players pulls out, even though the first set may have been completed by that point.

`for,tset,all,vwhole,set,cs,`***x***`,`***y***
: In the match, player 1 will score *x* sets and player 2 will score *y* sets.

`for,tset,all,vwhole,set,ah(over|under),`***n***
: In the match, the total number of sets will be over/under handicap code *n*.

`for,tset,all,vwhole,game,ah(over|under),`***n***
: In the match, the total number of games will be over/under handicap code *n*.

`for,tset,2,vwhole,game,ah(over|under),`***n***
: In the second set, the total number of games will be over/under handicap code *n*.

`for,tset,all,vwhole,set,ah,(p1|p2),`***n***
: In the match, the specified player will win more sets than their opponent, after a handicap code *n* has been applied to the result.

`for,tset,all,vwhole,game,ah,(p1|p2),`***n***
: In the match, the specified player will win more games than their opponent, after a handicap code *n* has been applied to the result.

### *7.2.2 Bet types with time periods*

Some non-tennis bet types will also have a time period indicator, of the form:

```
for,time_period,rest_of_bet_type
```

where *time_period* is one of:

`tall`, `"tp,all"`
: The bet applies to all time periods, including any overtime and penalty shootouts if appropriate.

`treg`, `"tp,reg"`
: The bet applies to regular time only.

`tp1`**,** `tp2`**,** `tp3`**, etc., or** `"tp,1"`**,** `"tp,2"`**,** `"tp,3"`**, etc.**

The bet applies to period 1 (or 2, 3 respectively) only.

`th1`**,** `th2`

The bet applies to the first (or second) half only.

`tet`**,** `tot`

The bet applies to extra time or overtime only.

`tq1`**,** `tq2`**,** `tq3`**,** `tq4`

The bet applies to the indicated quarter only.

`"tinnings,all"`**,** `"tinnings,N"` **(baseball)**

The bet applies to the end of indicated Inning, which can be a number `N`, or `all` to denote the entire match including overtime and penalties.

### *7.2.3 BF any other unquoted*

`for,aou,h,`***n***

BF any other unquoted (Home wins). The number *n* is the highest value for which (x,x) is in the CS market.

`for,aou,a,`***n***

BF any other unquoted (Away wins). The number *n* is the highest value for which (x,x) is in the CS market.

`for,aou,d,`***n***

BF any other unquoted (Draw). The number *n* is the highest value for which (x,x) is in the CS market.

## 7.3 List of Bet Statuses

The bet status. This is currently one of:

`done`

The bet was accepted by the bookie.

`error,`***reason***

The bet was not placed due to technical reasons.

`failed,`***reason***

The bet was not accepted by the bookie.

`danger`

The bet was provisionally accepted, but the bookie may void it in the near future.

`unknown`

We don't know whether the bet was accepted or not.

`order`

The bet has been placed as an order on an exchange, but has not yet been matched.

`finalising`

We have requested than an exchange order be cancelled, but don't yet have confirmation from the exchange that is has been cancelled.

`tracking`

The bet was probably accepted. This is an intermediate status that strongly implies the bet was accepted, but we don't yet know for sure, and we are unlikely to have a bookie bet id, a price or a stake for the bet.

`preparing`

The bet is being prepared for submission to the bookmarker website. The wanted stake of the bet is defined in terms of the order currency, at this point.

`placing`

: Bet is currently being placed; no response recorded from bookie. When the bet has transitioned from `preparing` to `placing` status, the `want_stake` will be converted from order currency into bookmaker account currency.

`tracking_error`

: There was an error during placement, and we're trying to automatically find the bet on the bet list.

`dangerorder`

: An open order on an exchange which is in danger state.

`settled`

: The bet is settled and the event closed, so it doesn't affect our position any more.

`void`

: The bet has been voided after being accepted.

## 7.4 List of Order Statuses

The order status. This is currently one of:

`open`

: can still place more bets.

`pending`

: order is closed, but has open bets (i.e. still placing).

`failed`

: order is closed, and all its bets failed to be placed (or no bets placed at all).

`partial_void`

: order is closed, placed at least one bet, at least one bet is voided.

`full_void`

: order is closed, placed at least one bet and all bets are voided.

`done`

: order is closed, placed at least one bet, none voided, some bets are done but not all bets are settled.

`reconciled`

: order is closed, placed at least one bet settled all bets, and all bets are reconciled (i.e. the profit/loss reported by the bookmaker matches the profit/loss MollyBet API expects).

`settled`

: same as above, but at least one bet is not reconciled.

## 7.5 List of Order Log Categories

The order log message category. This is currently one of:

`account_selection`

: Messages about accounts being added and removed.

`placement_summary`

: A summary of the placements made during a round. Omitted if no placements were made.

`round_summary`

: A detailed summary of the actions taken during a placement round. This includes error messages for rounds where no placement was made.

`status_update`

: Status updates for individual bets. Also includes the overall outcome of the order.

## 7.6 Glossary

***Offer***

A price offered by a bookie on a event it is possible to bet on. It may contain the minimum and maximum stake available to bet, if this information is readily available. The price is not related to a specific bookie account but it is generic for the whole bookie.

***PMM***

The specific price and minimum and maximum stake on a bookie account, requested opening a betslip. Requesting PMMs generates calls on the remote bookie/exchanges in order to get the most up-to-date information to bet.

***Future Fulfilment***

An order to place bets on a specific event at a minimum price, which MollyBet will keep open for a requested time: if the price requested is not readily available MollyBet will watch the market until the conditions for betting are fulfilled.

***Balance***

The cumulative profit and loss on a MollyBet account, together with other transfers, expenses and other monetary movements. It is expressed in a currency that is fixed at account creation.

***Credit Limit***

The amount of credit that is available for betting, usually the account balance but it can be extended according to customers' agreements with their own agents. The limit is usually tied to a single MollyBet account; additionally, if a set of accounts is grouped together, the group may have a global credit limit too.

***Effective Price***

The price of a bet net of commission, re-expressed as a decimal price.

***Placement Round***

An iteration of placement attempt, where MollyBet analyses the market and attempts to place at the best price available. If some of the requested stake couldn't be placed at the first iteration further placement rounds may follow.

***Settlement***

After a sport event is finished, the attribution of a profit or loss to all the bets placed on that event. The amount shall be then compared to what the bookmakers declare in reconciliation.

***Reconciliation***

The comparison of the profit or loss estimated in settlement with what the bookmaker return. In case there is a mismatch the bets are considered *not reconciled* and investigated.

***Open Stake***

The stake for placed bet that have not been settled yet.

***Blacklist***

A temporary suspension of a bookmaker account, when the system detects that using it may lead to problem (misbehaviour or bad integration of the bookie account with the rest of the system, or that there are likely conditions for the bets to be voided).

# 8 Frequently Asked Questions

**Q: Why are my bets still changing even after my order expired (or was cancelled)?**

MollyBet only controls placement, the bookie controls the outcome of already-placed bets. That means we will stop placing new bets as soon as the order expires or is cancelled but already-placed bets cannot be un-placed.

The exception are exchange orders which have not yet been matched. Unmatched stake is taken down from the exchange as soon as the MollyBet order expires.

**Q: When does MollyBet put my order on an exchange, and at what price?**

- If at the first placement round no bookie offers are found that match your order criteria, we will attempt to put the entirety of the stake on an exchange.
- If at the first placement round no bookie offers are found that match your order criteria, but we detect that bookie offers may soon become available, then we will attempt to put half of the stake on an exchange.

The price will be the worst (most conservative) of the best prices available on any exchange.

**Q: I've reached my credit limit. What can I do?**

Talk to your agent.

**Q: How can I withdraw my funds? I want to withdraw my funds!**

Talk to your agent.

**Q: Why does my async channel keep disconnecting?**

To some extent this depends on the internet connection between your client and the MollyBet servers.

Alternatively, MollyBet may be forcing the connection to close because your client is too slow to read from the async channel.

If the server is unable to send data immediately over the connection, the data is temporarily queued to give your client the opportunity to catch up. If the client is too slow, however, the size of this queue increases indefinitely, and the client starts receiving more and more outdated information. When MollyBet detects a client is lagging significantly, it shuts down the connection.

Two criteria can trigger this behaviour:

- there are more than 16,000 messages pending in the connection queue, or
- the oldest item in the queue is more than 5 minutes old.

**Q: Where are my MollyBet Accounting data from X months ago?**

Due to the volume of data, detailed accounts are stored for about 3 months. After that you can see summarised statements using /statements.

**Q: When does my MollyBet balance change?**

A bet placement request will have immediate effect on the MollyBet balance - the balance is updated by subtracting the stake needed for the bet. Of course this needs to be extremely fast, as we don't want to miss the requested price.

**Q: I'm trying to register to X events, but MollyBet won't let me. Why?**

At this time, the default limit is 500 simultaneous events *per MollyBet group* at any one time.

As registration limits are enforced on a group by group basis, a different user (or another connection by the same user) may be using some of the quota.

Two connections registering to the same event both count towards the limit.

**Q: Why do my bookies/account usernames look like "_ab3c_"?**

This means that your agent has chosen to hide that information from you. While the codes are randomly generated for your accounts, they will be consistent.

**Q: Why do I get the same data over HTTP and async when I create an order/betslip?**

The TCP protocol guarantees that data will be received in the same order it was sent *over the same connection*, however the HTTP and async run over two different TCP connections. In other words, there is no way to guarantee that the HTTP call will return before or after the async message.

**Q: Why do I receive an insecure (self-signed) certificate when connecting to the API?**

If you get a self-signed certificate it probably means that your client HTTP/TLS libraries do not support the Server Name Identification (SNI) protocol extension, which is required in order to connect to the API.

## 9 Legal

Your continued use of MollyBet API is governed by the legal agreements you can find at this link.