Swarm Public API Documentation

Release 1.0

BetConstruct (d.gasparian)

April 25, 2017

CHAPTER

ONE

CLIENT DEVELOPMENT - CONNECTING AND GETTING DATA

Developing clients for Swarm is easy. In fact, Swarm was designed to take the lion's share of the work on its shoulders, so you don't have to worry about anything other than specifying the data that you need.

This guide is meant to be a reference for client developers. It is written to be technology-agnostic, so whether you're using Javascript or QT, this is the best place to start coding your Swarm client application.

1.1 Long poll and Websockets

All of the data interchange happening between your client application and Swarm must take place via HTTP.

Swarm supports two transport methods over HTTP: Long poll and Websockets.

You have to decide on the HTTP transport you prefer your client application to use, depending on the requirements and technical constraints. It should be noted that when both technologies are available, using Websocket is preferred both because it's more straightforward to program, and because it's easier on the server resources.

Tip: Modern web browsers support both Long poll and Websockets. If the client is a web application, it makes sense for the client to be programmed for both: use Websockets when they are available, and gradually degrade to using Long poll in older browsers.

Swarm provides transport method abstraction to the client in a way that it allows the underlying client logic between Long poll and Websockets to remain pretty much the same (if coded right).

Working via Long polls means that the client should occasionally make a `whats_up()` request to Swarm. Working via Websockets means that all the communication between the client and Swarm will happen through a single Websocket channel.

Note: Swarm supports *Cross-Origin Resource Sharing (CORS)*. Your `OPTIONS` preflight requests will be properly served.

Note: Swarm supports both regular HTTP and secure HTTPS. Using HTTPS is recommended for apparent security advantages. Note that secure websocket connections are specified by protocol *wss://* rather than ws://. Swarm endpoint URL will be provided by BetConstruct during integration process.

1.2 Data

Swarm's data interchange format is JSON.

All of the data received from Swarm is formatted as JSON.

All of the commands and parameters sent to Swarm are expected to be JSON as well.

> **Warning:** The JSON format specifies that keys and string values be enclosed in **double** quotes *("like this", and not 'like this')*. Although some liberal parsers do tolerate single quotes, Swarm does not. Always enclose your keys and string values in double quotes.

Tip: Just as whitespace is not significant in JSON, it is not significant for Swarm. To transport less data across network, Swarm strips all whitespace from the data it sends to the client. For the same purpose, it's advised that the client does the same.

Note: For the sake of readability and clarity, all example JSON snippets in this guide are properly formatted and whitespaced.

1.3 Commanding Swarm

Swarm commands are issued either using regular HTTP POST requests (when using Long poll), or through a Websocket channel (see Long poll and Websockets).

The general format of the command in both cases is the following:

```
{
    "command": [command],
    "params": {
        [parameter name]: [parameter value]
    }
}
```

Where *[command]* is one of the following:

request_session Request a new session

get Get data, and optionally subscribe to data changes

whats_up Poll for changes to subscriptions (Long poll only)

unsubscribe Unsubscribe from an earlier subscription

For commands requiring no parameters, "params" may be omitted entirely:

```
{
    "command": [command]
}
```

The format of the response:

```
{
    "code": 0,
    "data": {
        ...
    },
    "rid": 0,
    "msg": "...",
}
```

Where:

code Error code of response. See *Appendix A: Swarm Error Codes* for list of values.

data Actual data payload returned.

rid Client Request ID. optional

msg Descriptive message in case of errors. optional

1.3.1 Client Request IDs

Due to the asynchronous nature of Swarm, using a persistent communication channel like Websocket makes it difficult to map incoming responses from Swarm to their respective requests sent earlier. Consider the following example scenario:

```
// <- RESPONSE
{
    "code": 0,
    "data": {
        "data": {
            "game": {
                "100": {
                    "name": "India - Delhi Senior Division (LIVE)"
                }
            }
        }
    }
}

// <- RESPONSE
{
    "code": 0,
    "data": {
        "data": {
            "game": {
                "101": {
                    "name": "ISRAEL - YOUTH CUP (LIVE)",
                }
            }
        }
    }
}

// -> REQUEST
{
    "command": "get",
    "params": {
        "source": "betting",
        "what": {
            "game": ["name"]
        },
        "where": {
            "game": {"id": 101}
        }
    }
}

// -> REQUEST
{
    "command": "get",
    "params": {
```

```
        "source": "betting",
        "what": {
            "game": ["name"]
        },
        "where": {
            "game": {"id": 100}
        }
    }
}
```

The example shows 2 subsequent requests with their respective responses. However, the order by which the responses have arrived is different from the order by which the requests were issued. There is therefore a challenge to properly map each arriving response to the request that yielded it.

To solve this challenge for the client, Swarm accepts an optional parameter named `rid`: Request ID. You can pass along an arbitrary value for a request parameter named `rid`, and be sure that the response to that particular request will contain the value you passed:

```
// <- RESPONSE
{
    "code": 0,
    "rid": 1,
    "data": {
        "data": {
            "game": {
                "101": {
                    "name": "India - Delhi Senior Division (LIVE)",
                }
            }
        }
    }
}

// -> REQUEST
{
    "command": "get",
    "rid": 1,
    "params": {
        "source": "betting",
        "what": {
            "game": ["name"]
        },
        "where": {
            "game": {"id": 101}
        }
    }
}
```

Note: Data updates for subscriptions never contain a `rid` or will have a value of 0, since they are not a direct response to any issued request. The response to the initial `get` command that subscribed to the data will, however, contain any Request ID passed to it.

1.4 The Big Picture

To outline very briefly, here is the list of steps the typical client does to communicate with Swarm:

1. Request a session
2. Use the session with every subsequent request to get data or to subscribe to data changes
3. Monitor arriving updates for subscribed data, and populate it in the application
4. Unsubscribe from data subscriptions, when the particular set of data is no longer needed
5. In case session is reported dead during any of the regular calls, or when the Websocket connection breaks down, request another session and resubscribe to all the data previously subscribed to

Performing each of these steps is painstakingly easy. In fact, everything outlined in the above steps is achieved by issuing only the commands specified in the section Commanding Swarm.

Each of the five steps is covered extensively in the next sections.

1.5 Sessions

Swarm identifies each client by generating sessions and issuing session IDs (SIDs). Sessions are generated upon request by the client. Except for `request_session` itself, every command issued to Swarm requires a SID. This is natural – there is no other way for Swarm to be able to conveniently map the information that it has about your state (subscriptions, preferred language, etc) to your request.

While the general logic of session management is the same, there are subtle differences in how this logic is performed for Websocket and Long poll. These differences are specified in the following sections.

1.5.1 Requesting a Session

Since almost all Swarm commands require a SID, the application must acquire one before issuing further commands. This is done using `request_session()`:

Example request:

```
{
    "command": "request_session",
    "params": {
        "site_id": 1,
        "language": "arm",

        // optional
        "source": 1, // source field
        "terminal": 123 // terminal field
    }
}
```

`request_session` accepts following parameters:

site_id SiteID (also known as PartnerID) provided to specific partner by BetConstruct.

language Language to return data in. See *Appendix B: Language Codes* for list of possible values.

source Session origin/source, integer number to distinguish origin of session optional

terminal Terminal number (ID) where session originates from optional

As can be inferred from the signature of `request_session()`, language and site ID are bound to the session. Therefore, when switching the language, a new session must be acquired.

Example response:

```
{
    ...
    "data": {
        "sid": "424e8ec4-5b6d-42d3-888e-2858b23d35a2"
    }
}
```

where:

sid Assigned Session ID to use for subsequent requests.

Note: When using Websocket, in case the connection is lost, a new session must be requested.

Swarm does not support session recovery or transfer. If the server responds with an *Invalid Session* error, the client must request another session and resubscribe to the existing subscriptions.

1.5.2 Passing the Session ID for Long Poll

Unlike Websocket, where the client has a single persistent channel with Swarm through which it posts requests and receives responses, Long poll requests are independent from each other, and each Swarm request is a separate HTTP request of its own. In order for Swarm to be able to map these requests to their respective sessions, a Session ID must be sent with each request.

Swarm expects that the SID for Long poll requests is sent in an HTTP header named `swarm-session`:

Example header:

```
swarm-session: 424e8ec4-5b6d-42d3-888e-2858b23d35a2
```

Failure to include `swarm-session` in the headers when issuing a command other than `request_session` to Swarm will yield HTTP error *401 Forbidden*.

1.5.3 Passing the Session ID for Websocket

There is no need to pass SID when using Websocket. After requesting a session via a Websocket channel, Swarm will automatically treat future commands coming via that channel as belonging to the requested SID.

1.5.4 Keeping session alive

In case of Websocket connections sessions are kept alive as long as connection is open.

In Long Poll case session will be cleaned up after some timeout in there's no activity over it. So if for some reason you're not issuing any commands for a long time but don't want to lose current SWARM session its possible to keep it alive by issuing `whats_up()` command periodically.

Tip: The default SWARM session timeout setting is 3 min.

1.5.5 Removing a session

It would be a good practice to issue manual session removal when SWARM services are no longer needed.

```
{
    "command": "remove_session"
}
```

1.6 Authentication related commands

So first of all before any user specific operation is done - user should be authenticated.

This is there login commands comes in help. Without it all further commands (for example placing bet) with fail with "Not authenticated" error. There are 2 ways of logging in: using username/password and using user id/auth token (which can be obtained after first login).

1.6.1 Logging in using username and password

Example request:

```
{
    "command": "login",
    "params": {
        "username": "testuser",
        "password": "hispassword"
    }
}
```

where:

username User ID.

password User password. Plaintext.

Successful response example:

```
{
    "code": 0,
    "data": {
        "auth_token": "test_token_228",
        "user_id": 228
    }
}
```

As you can see `code` indicates success, and `user_id` and `auth_token` are included in response.

you may consider saving received `user_id` and `auth_token` to use them later for logging in without providing a password, see next section for detailed description.

Error response example:

```
{
    "code": 12,
    "data": "login error (1002)",
    "msg": "Invalid credentials"
}
```

`code` and `msg` here are self-descriptive. `data` contains error number forwarded from backend. See *Appendix D: Swarm Error Codes* for more information.

1.6.2 Logging in using user id and auth token

Example request:

```
{
    "command": "restore_login",
    "params": {
        "user_id": 11111,
        "auth_token": "some-random-authToken-12345678"
    }
}
```

where:

user_id User ID in the system.

auth_token Valid AuthToken for user.

Successful response example:

```
{
    "code": 0,
    "data": {
        "auth_token": "test_token_228",
        "user_id": 228
    }
}
```

As you can see `code` indicates success, and `user_id` and `auth_token` are echoed back.

Error response example:

```
{
    "code": 12,
    "data": "login error (1002)",
    "msg": "Invalid credentials"
}
```

Response is quite the same as when logging in with username/password. `code` and `msg` here are self-descriptive. `data` contains error number forwarded from backend.

1.6.3 Logging out

At some point it might be needed to `logout` current user.

```
{
    "command": "logout",
    "params": {}
}
```

Logout response:

```
{
    "code": 0,
    "data": null
}
```

1.7 User registration

1.7.1 Creating new user

Example request:

```
{
    "command": "register_user",
    "params":{
        "user_info": {
            "username": "mynewusername",
            "password": "mypassword",
            "first_name": "FirstName",
            "last_name": "LastName",
            "middle_name": "MiddleName",
            "gender": "M",
            "city": "London",
            "birth_date": "1964-01-01",
            "address": "Address 123",
            "country_code": "AF",
            "email": "myemailaddress@mail.com",
            "phone": "93123456",
            "currency_name": "USD",
            "doc_number": "",
            "site_id": "4",
            "promo_code": "789",
            "security_question": "What was the name of your first pet?",
            "security_answer": "computer",
            "notify_via_email": true,
            "notify_via_sms": null
            "captcha_text": "captchaText"
        }
    }
}
```

where:

username User login.

password User password.

first_name First name.

last_name Last name.

middle_name Middle name.

gender Gender (“M” or “F”).

city City name.

birth_date Birth date in YYYY-MM-DD format.

address Postal address.

country_code 2-letter country code.

email User’s email address.

phone User’s phone number.

currency_name Currency 3-letter code.

doc_number Identification document number.

site_id Partner id.

promo_code Promotional code.

security_question Security question.

security_answer Answer to security question.

notify_via_email Whether user want to get emails with news, promotions, etc (null or true).

notify_via_sms Whether user want to get SMS with news, promotions, etc (null or true).

captcha_text Captcha text (this field is required if the functional enabled for the partner, see also Command get_captcha_url).

Note: Most of the fields can be configured per partner to be mandatory or not and new fields can be added upon request.

Successful response example:

```
{

    "code":0,
        "data": {
        "result": "OK",
        "details": {
            "username": "mynewusername",
            "currency_name": "USD",
            "uid": "1174872847",
            "unique_id": "2465611"
            }
        }
}
```

Response field `result` indicates registration process success.

Some possible `result` values are:

OK Registration was successful.

-1013 Password is too short.

-1012 Phone number is not correct.

-1127 There's already another user with same phone number.

-1014 Failed to send verification SMS (if phone number SMS verification is enabled for partner).

-1118 Username already exists.

-1119 Username already exists.

-1010 Password cannot be the same as login.

-1123 There's already another user with same document number.

1.8 User profile related commands

This chapter describes user profile related commands. Before executing any of those commands user must be logged in using the login command.

1.8.1 Command update_user

Update user info. Login required.

Example request:

```
{
    "command": "update_user",
    "params": {
        'user_info': {
                            'password': 'user_current_password',
                            <field1>: <new_field1_value>,
                            <field2>: <new_field2_value>,
                            ..........................
        }
        }
}
```

where:

password User current password [mandatory].

<field1>, <new_field1_value> One of the following.

```
{
    "city": "New_name",
    "first_name": "New_name",
    "user_id": "2081621",
    "name": "test test",
    "doc_number": "546541",
    "country_id": "2265010",
    "sex": "M",
    "currency_id": "971",
    "phone": "+374999999",
    "second_name": " ",
    "sur_name": "test",
    "country_code": "AE",
    "address": "nnnnn",
    "birth_date": "1989-10-02",
    "balance": "1.5265",
    "email": "example@mailhost.com"
}
```

Successful response example:

```
{
    "code": 0,
    "data": {
        "result": 0,
        "details": {}
    }
}
```

1.8.2 Command update_user_password

Command to update user's password. This command requires the logged in session.

```
{
    "command": "update_user_password",
```

```
    "params": {
        "password": "CurrentPassword",
        "new_password": "NeWPassword"
    }
}
```

where:

password User's current password.

new_password New password.

Successful response example:

```
{
    "rid": 10003,
    "code": 0,
    "data": {
        "auth_token": "9e132e8d4e55a3fb35caaa399e68913a26c80fbb11ac332da95fd1afebfb3d4e"
    }
}
```

1.8.3 Command get_user

Returns logged in user's profile. This command requires the logged in session.

Example request:

```
{
    "command": "get_user",
    "params": {
        }
}
```

Successful response example:

```
{
    "rid": 10003,
"code": 0,
"data": {
    "city": "Yerevan",
    "first_name": "test",
    "user_id": "2081621",
    "username": "test test",
    "doc_number": "546541",
    "sex": "M",
    "currency_id": "971",
    "phone": "+374999999",
    "second_name": " ",
    "sur_name": "test",
    "country_code": "AE",
    "address": "nnnnn",
    "birth_date": "1989-10-02",
    "balance": "1.5265",
    "email": "example@mailhost.com"
    }
}
```

1.8.4 Command add_user_message

Add user feedback message. Login required.

Example request:

```
{
    "command": "add_user_message",
    "params": {
        "subject":"Subject of the message",
        "body":"Some Text"
    }
}
```

Successful response example:

```
{
    "rid": 10003,
    "code": 0,
    "data": {
        "result": 0,
        "details": {}
    }
}
```

1.8.5 Command user_messages

Returns user's messages sent to or received from the support service. This command requires the logged in session.

Example request:

```
{
    "command": "user_messages",
    "params": {
        "where" : {
            "type" : 1
        }
    }
}
```

where:

type 0 - incoming messages, 1 - outgoing messages. [optional] - if absent show all types of messages.

Successful response example:

```
{
    "rid": 10003,
    "code": 0,
    "data": {
        "messages": [
            {
                "body": "Some Text",
                "checked": "0",
                "thread_id": "1777520",
                "date": "1429096530",
                "id": "1254322033",
                "subject": "Subject of the message"
            },
```

```
            {
                "body": "Another text",
                "checked": "0",
                "thread_id": "1777508",
                "date": "1429095775",
                "id": "1254306789",
                "subject": "Another subject of the message"
            }
        ]
    }
}
```

1.8.6 Command read_user_message

Marks user message as seen. This command requires the logged in session.

Example request:

```
{
    "command": "read_user_message",
    "params": {"message_id" : 1175376747}
}
```

where:

message_id Message unique id.

Successful response example:

```
{
    "rid": 10003,
    "code": 0,
    "data": {
        "result": 0,
        "details": {}
    }
}
```

1.8.7 Command user_limits

Returns user's limits like maximum deposit amount per single deposit/day/week/month. The amounts are in user's currency. This command requires the logged in session.

Example request:

```
{
    "command": "user_limits",
    "params": {
        "type": "deposit"
    }
}
```

where:

type Limit type [Currently supports only "deposit"].

Successful response example:

```
{
    "rid": 10003,
    "code": 0,
    "data": {
    "result": 0,
    "details": {
        "max_month_deposit": 1000000.0,
        "max_single_deposit": 1000.0,
        "max_day_deposit": 5000.0,
        "max_week_deposit": 100000.0
        }
    }
}
```


1.8.8 Command set_user_limits

Sets user's limits like maximum deposit amount and self-exclusion. This command requires the logged in session.

Example request:

Sets user's maximum deposit amount per single deposit/day/week/month. The amounts are in user's currency.


```
{
    "command": "set_user_limits",
    "params": {
        "type": "deposit",
        "limits": [
            {
                "deposit_limit": 5000,
                "period_type": 2,
                "period": 1
            }
        ]
    }
}
```


where:

deposit_limit Amount of deposit.

period_type time period type (2: Day, 3: Week, 4: Month, 5: Year)

period count of deposits per time period

Successful response example:


```
{
    "code": 0,
    "rid": "10003",
    "data": {
        "result": 0,
        "data": {}
    }
}
```


Example request:

Sets user's self-exclusion for a duration of days/weeks/months.

```
{
    "command": "set_user_limits",
    "params": {
        "type": "self-exclusion",
        "limits": {
            "months": "6",
        }
    }
}
```

where:

months time_period (months, years)

Successful response example:

```
{
    "code": 0,
    "rid": "10003",
    "data": {
        "result": 0,
        "data": {}
    }
}
```

1.8.9 Command upload_image

Uploads users profile image. This command requires the logged in session.

Example request:

```
{
    "command": "upload_image",
    "params": {"image_data" : "iVBORw0KGgoAAAANSUhEUgAAABgAAAAYCAYAAADgdz34AAAAGXRFWHRTb2Z0d2FyZQBBZ
}
```

where:

image_data Base64 encoded image.

Successful response example:

```
{
    "rid": 10003,
    "code": 0,
    "data": {
        "result": 0
    }
}
```

1.9 Balance related commands

This chapter describes user's balance specific commands. All commands in this section must be called within logged in user session.

1.9.1 Command deposit

Process money deposit.

Example request:

```
{
    "command": "deposit",
    "params": {
        'amount': 20,
        'service': 'eblontransfer',
        'payer': {
            'customer_id': '5D1780346',
            'customer_password': 'passwd',
            }
        }
    }
}
```

where:

amount Amount of deposit.

service Service type (skrill, webmoney, moneta, ecocard, qiwi, ...)

payer parameters for specific service.

Successful response example:

```
{
    "rid": 10003,
    "code": 0,
    "data": {
        "result": 0
    }
}
```

1.9.2 Command get_deposit_status

Example request:

Returns the status for the user's deposit request.

```
{
    "command": "get_deposit_status",
    "params": {
        "service": "blockio",
        "transaction_id": 666
    }
}
```

where:

service Service type.

transaction_id Deposit order TransactionID [optional].

Successful response example:

```
{
    "rid": 10003,
    "code": 0,
    "data": {
        "Result": 0
    }
}
```

1.9.3 Command withdraw

Withdraw money from specified service.

Example request:

```
{
    "command": "withdraw",
    "params": {
        'amount': 0.01,
        'service': "skrill",
        'payee': {
            'email': 'user@mailhost.com',
            'name': 'User name'
        }
    }
}
```

where:

amount Amount of deposit.

service Service type (skrill, webmoney, moneta, ecocard, qiwi, ...)

payee parameters for specific service.

Successful response example:

```
{
    "rid": 10003,
    "code": 0,
    "data": {
        "Result": 0
    }
}
```

1.9.4 Command get_withdrawals

Returns user pending withdrawal requests.

Example request:

```
{
    "command": "get_withdrawals",
    "params": {
          "from_date": 123123123
          "to_date": 123123456
    }
}
```

where:

from_date From Date timestamp parameter [optional].

to_date To Date timestamp parameter [optional].

Successful response example:

```
{
    "rid": 10003,
    "code": 0,
    "data": {
        "withdrawal_requests": {
            "request": [
                {
                    "date": "2014-04-29 11:51:01",
                    "status": "1",
                    "amount": "0.0216",
                    "id": "851805764",
                    "name": ""
                },
                {
                    "date": "2015-02-09 14:50:31",
                    "status": "0",
                    "amount": "0.1",
                    "id": "1164222533",
                    "name": ""
                }
            ]
        },
    }
}
```

1.9.5 Command withdraw_cancel

Cancels pending withdrawal request.

Example request:

```
{
    "command": "withdraw_cancel",
    "params": {
        "id": 12345
    }
}
```

where:

id pending withdrawal id.

Successful response example:

```
{
    "rid": 10003,
    "code": 0,
    "data": {
        "result" : 0
    }
}
```

1.9.6 Command payment_services

Returns available payment services for current site/user-currency.

Example request:

```
{
    "command": "payment_services",
    "params": {}
}
```

Response:

```
{
    "rid": 10003,
"code": 0,
"data": {
    "deposit": [
        "skrill",
        "moneybookers",
        "skrill1tap",
        "moneybookers1tap",
        "webmoney",
        "moneta",
        "moneta.ru",
        "wirecard",
        "netellernew",
        "astropay",
        "yandex",
        "ecocard",
        "ukash",
        "DengiOnline_LiqPay",
        "DengiOnline_EasyPay"
    ],
    "withdraw": [
        "skrill",
        "moneybookers",
        "skrill1tap",
        "moneybookers1tap",
        "webmoney",
        "moneta",
        "moneta.ru",
        "netellernew",
        "astropay",
        "yandex",
        "ecocard"
    ]
    }

}
```

1.9.7 Command balance_history

Returns logged in user's balance history, e.g. deposit/withdraw transactions.

Example request:

```
{
    "command": "balance_history",
    "params": {
        "where": {
            "from_date": 1430251200,
            "to_date": 1430424000
            }
        "product":"Casino"
    }
}
```

where:

from_date From Date timestamp parameter.

to_date To Date timestamp parameter.

product Product type "Sport" or "Casino" optional default one is "Sport".

Successful response example:

```
{
    "rid": 10003,
    "code": 0,
    "data": {
        "history": [
                {
                "amount":67.0,
                "bet_id":220964075,
                "transaction_id":459460628,
                "operation":1,
                "operation_name":"Increasing the winning",
                "balance":8398.0,
                "date_time":1476557879,
                "game":"SportsBook",
                "product_category":1
                },
            {
                "amount":-50.0,
                "bet_id":220930174,
                "transaction_id":459392000,
                "operation":0,
                "operation_name":"Bet",
                "balance":8331.0,
                "date_time":1476557136,
                "game":"SportsBook",
                "product_category":1
                },
        ]
    }
}
```

where

operation possible values see *Appendix D: Swarm Error Codes* section 7.4 Balance History operation values.

1.10 Bonus related commands

This chapter includes commands related to promo bonuses available in Spring Platform. Bonuses are enabled per partner by setting appropriate configurations in the Spring platform. Contact with your account manager for this purpose.

1.10.1 Command get_bonus_details

Get client bonus details. Login required.

Example request:

```
{
  "command": "get_bonus_details",
  "params": {
      "free_bonuses" : true
  }
}
```

where:

free_bonuses if true: get sportsbook wagering available bonuses, if false: get casino available bonuses.

Successful response example:

```
{
    "code":0,
    "data":{
      "bonuses":[
        {
        "id":41279,
        "partner_bonus_id":217,
        "source":0,
        "name":"Bonus name",
        "description":"Bonus detailed description",
        "start_date":1475442000,
        "end_date":1475528400,
        "client_bonus_expiration_date":14761080
        "expiration_days":7,
        "wagering_factor":0,
        "can_accept":false, // whether user can claim the bonus or not
        "bonus_type":6,
        "amount":2000.0,
        "acceptance_type":2,// 0: not claimed, 1: claimed but have not made deposit, 2: deposit
        "result_type":0
        },
        ...
      ]
    }
}
```

1.10.2 Command get_freebets_for_betslip

Get available freebets for particular combined betslip. Call the command when “has_free_bets”:true, field exists in user *profile*!!! Login required.

Example request:

```
{
  "command":"get_freebets_for_betslip",
  "params":{
      "type":1,
      "source":"1",
      "is_offer":0,
      "mode":0,
      "each_way":false,
      "bets":[
        {
          "event_id":129455310,
          "price":2.3
        }
      ],
      "is_live":true
  },
}
```

where:

params should be same data model as in do_bet command

Successful response example:

```
{
  "code":0,
  "data":{
    "result":0,
    "details":[
      {
        "$id":"1",
        "id":42386,
        "acceptance_type":2,
        "acceptance_date":"2016-10-05T12:34:50.95+04:00",
        "client_id":11525981,
        "count":1,
        "partner_bonus_id":221,
        "result_type":0,
        "name":"test_name",
        "description":"test_description"
        "expiration_days":7,
        "start_date":"2016-10-03T20:00:00+00:00",
        "end_date":"2016-10-29T20:00:00+00:00",
        "is_visible_to_all":false,
        "amount":100.0,
        "client_bonus_expiration_date":"2016-10-12T12:34:34.45+04:00",
        "bonus_type":6,
        "partner_id":32
        "source":0
      }
    ]
  },
}
```

1.10.3 Command claim_bonus

Process claim request for particular bonus (can_accept should be true in get_bonus_details). Login required.

Example request:

```
{
  "command": "claim_bonus",
  "params": {
      "bonus_id" : 60
  }
}
```

where:

bonus_id Bonus id.

Successful response example:

```
{
    "code":0,
    "data":{
      "result": 0
    }
}
```

1.10.4 Command cancel_bonus

Process cancel request for particular bonus (acceptance_type should be 1 in get_bonus_details). Login required.

Example request:

```
{
  "command": "cancel_bonus",
  "params": {
      "bonus_id" : 60
  }
}
```

where:

bonus_id Bonus id.

Successful response example:

```
{
    "code":0,
    "data":{
      "result": 0
    }
}
```

1.11 Other commands

This chapter includes commands not related to any of other chapter.

1.11.1 Command get_tournaments

Get tournaments info for specified product type if there are any.

Example request:

```
{
    "command": "get_tournaments",
    "params": {
        "product_type" : 1
        }
    }
```

Successful response example:

```
{
    "rid": 10003,
    "code": 0,
    "data": {
        "start_time": "2015-04-29 15:00",
        "entrance_fee": "0.05C",
        "prize": "10.00C",
        "id": 10594,
        "name": "Mini Backg"

    }
}
```

1.11.2 Command get_victorina_info

Returns victorina events.

Example request:

```
{
    "command": "get_victorina_info",
    "params": {
        "day": 0
        }
    }
```

Successful response example:

```
{
    "rid": 10003,
    "code": 0,
    "data": {
        "victorinas":{
            "3711":{
                "1261857145":{
                    "competition_name":"Football. Spain - Primera Division (Liga BBVA)",
                    "sport_alias":"Soccer",
                    "sport_name":"Football",
                    "events":{
                        "p2":{
                            "event_id":"1261857885",
                            "k":"3.20"
                        },
                        "x":{
                            "event_id":"1261857884",
                            "k":"3.15"
                        },
                        "p1":{
                            "event_id":"1261857883",
```

```
                        "k":"2.34"
                    }
                }
            },
            "game_name":"Elche - Deportivo La CoruÃ±a",
            "have_bet":false,
            "game_start_date":"2015-04-30 00:00:00",
            "order":3
            }
        }
    }
}
```

1.11.3 Command do_bet_victorina

Place a victorina bet.

Example request:

```
{
    "command": "do_bet_victorina",
    "params": {
        "victorina_id":"3711",
        "selections":[{"1261857140":1261857717},{"1261857145":1261857884}]}}
        }
    }
```

Successful response example:

```
{
    "rid": 10003,
    "code": 0,
    "data": {
        "code" : 0,
        "details": null
    }
}
```

1.11.4 Command user_feedback

Adds a feedback message from logged in user.

```
{
    "command": "user_feedback",
    "params": {
        'email': 'example@mailhost.com' [optional],
        'body': 'some feedback',
        }
}
```

Successful response example:

```
{
    "rid": 10003,
    "code": 0,
    "data": {
        "result": 0
```

```
    }
}
```

1.11.5 Command get_result_games

Returns results for the games that match the provided values

```
{
    "command": "get_result_games",
    "params": {
        // optional
        "from_date": "2015-05-04",
        "to_date": "2015-05-05",
        "sport_id": 844,
        "region_id": 65537,
        "competition_id": 89378912,
        "live": 0
        }
}
```

Successful response example:

```
{
    "rid": 10003,
    "code": 0,
    "data": {
        "games": {
            "game": [
                {
                    "competition_name": "Football. Austria - VFV-Cup",
                    "scores1": "1:0",
                    "game_name": "FC Höchst - Golm FC Schruns",
                    "scores": "2:1",
                    "date": "2015-05-05 20:30:00",
                    "game_id": "1281431520",
                    "odd": "0",
                    "sport_id": "844"
                }
            ]
        }
    }
}
```

1.11.6 Command get_captcha_url

Get captcha url

Example request:

```
{
    "command": "get_captcha_url",
    "params": {}
}
```

Successful response example:

```
{
    "rid": 10003,
    "code": 0,
    "data": {
        "url": "http://cptca.betconstruct.com/captcha.php?key=13039d1b1b1584334fdd"
    }
}
```


1.11.7 Command forgot_password

Reset user password

Example request:


```
{
    "command": "forgot_password",
    "params": {
        "email":"myemailaddress@mail.com"
    }
}
```


Successful response example:


```
{
    "rid": 10003,
    "code": 0,
    "data": {
        "result": 0,
        "details": {}
    }
}
```


1.11.8 Command get_results

Returns results for the game with the provided game_id


```
{
    "command": "get_results",
    "params": {
        "game_id": 1275634450
    }
}
```


Successful response example:


```
{
    "rid": 10003,
    "code": 0,
    "data": {
        "lines": {
            "line": [
                {
                  "line_name": "Total Goals: Even/Odd",
                  "events": {
                    "event_name": "Match Total: Even"
                  }
                },
```

```
                {
                  "line_name": "Asian Handicap (-1)",
                  "events": {
                    "event_name": "Asian Handicap 2(1)"
                  }
                }
            ]
        }
    }
```

1.11.9 Command partner.configs

Returns partner configuration specific fields

```
{
    "command": "get",
    "params": {
        "source": "partner.config",
        "what": {"partner": []}
    }
}
```

Successful response example:

```
{
    "code": 0,
    "data": {
    "partner": {
        "4": {
            "partner_id": 4,
            "is_cashout_prematch": 0,
            "is_cashout_live": 0,
            "currency": "USD"
            }
        }
    }
}
```

where:

partner_id Partner Id

is_cashout_prematch If 1 then prematch CashOut functionality enabled for partner, 0 otherwise

is_cashout_live If 1 then live CashOut functionality enabled for partner, 0 otherwise

currency Default currency set for partner

1.11.10 Command Accept terms and conditions

Before user's first deposit he needs to accept terms and conditions to be able to deposit.

This functional is configurable per partner in Spring platform. Contact with your account manager for this purpose. Login required.

```
{
    "command": "accept_terms_conditions",
```

```
    "params":{}
}
```

Successful response example:

```
{
    "code": 0,
    "data": {
       "result": 0,
       "details": {}
    }
}
```

1.11.11 Command cashOut

Process Bet's CashOut. Client needs to call the command for particular bet if "cash_out" field exist in "bet_history" call.

Example request:

```
{
    "command": "cashout",
    "params": {
        "bet_id":838198,
        "price":45
    }
}
```

Successful response example:

```
{
    "rid": 10003,
    "code": 0,
    "data": {
        "result":"Ok",
        "details": {
            "price":45.0
            }
        }
    }
}
```

1.12 Getting Data

Getting data is what Swarm is all about. It is also what Swarm excels at. In a way, Swarm could be described as a giant dynamic datastore. Like in ordinary database systems, you submit queries that it responds to with the resulting dataset. What makes Swarm different is that you won't have to poll the same query over again to get the dataset changes – you simply *subscribe* to the query, and any changes to the resulting dataset over time are automatically pushed to you. You only have to worry about properly populating the changes in your application.

Data in Swarm is organized into collections. These collections are referred to as *sources*. There are presently 3 sources in Swarm to get data from: *betting*, *user*, *messages*. **betting** contains all the betting data. **user** contains information about the authorized user. **messages** contains information about user messages.

A single record of data in Swarm is called a *node*. The nodes under *betting* are categorized into 6 hierarchic levels:

1. sport
2. region
3. competition
4. game
5. market
6. event

The nodes under *profile* are organized into 2 hierarchic levels:

1. profile
2. history

So, a node may be a sport node, or a competition node, or a profile node, or any of the other levels node. Each node has a set of *fields*. Obviously, the fields of data contained in a node of one level is very different from the fields of data contained in a node of a different level. Below is a comparison of sample data between a sport node and an event node.

Sport node:

```
{
    "id": 844,
    "name": "Soccer",
    "alias": "soccer",
    "order": 1
}
```

Event node:

```
{
    "id": 683678180,
    "name": "Handicap 1",
    "base": "1.5",
    "price": "1.57",
    "type": "Handicap1",
    "order": 3
}
```

The complete specification of data fields can be found in *Data*.

1.12.1 The get Command

The command for getting any data in Swarm is `get()`:

Example request:

```
{
    "command": "get",
    "params": {
        "source": "betting",
        "what": {"sport": []},
        "where": {},
        "subscribe": false
    }
}
```

The example above queries all sport nodes without any filtering conditions.

get is declarative in nature. Instead of specifying *how* to fetch the data, you specify *what* data you are interested in – `get` ensures that you get it. Mastering the declarative syntax of `get` queries is therefore one of the cornerstones of working with Swarm efficiently.

`get` accepts 4 parameters:

source Collection source

what Data to select

where Conditions to filter the selected data optional

subscribe Subscribe to the query optional

source

source defines the collection source to fetch data from. It can have one of the following values:

betting Information about all sports, competitions, games, and other betting data

user Information about the logged in user

messages Information about messages for logged in user

See also:

Data

For extensive specification of data collections and the data they contain, see *Data*.

what

what specifies the data to get out of the collection specified in source.

General syntax:

```
{
    'level': selector
}
```

level Level name

selector One of three possible field specifiers to select from the *level*

Selector syntax:

- `[]`: All fields for the nodes
- `['field_one', 'field_two', ...]`: Specific fields for the nodes
- `'@count'`: Number of matching nodes rather than nodes themselves

Note: If multiple levels are selected, Swarm automatically joins their data together.

Note: To use Outer Join instead of regular join, replace regular square brackets (`[...]`) with double square brackets (`[[...]]`) when specifying the selector.

See also:

`join`

where

where defines the filter to apply to the selected data. It is an optional parameter.

General syntax:

```
{
    'level': {
        'field': criteria
    }
}
```

level Level name

field Field name for the given *level* to filter for

criteria Criteria for *field* value to be filtered with. Criteria can be either *simple* or *complex*.

Simple criteria syntax:

```
{
    'level': {
        'field': value
    }
}
```

value Match against *value* using simple, type-significant direct comparison

Complex criteria syntax:

```
{
    'level': {
        'field': {
            'operator': operand
        }
    }
}
```

operator Operator to filter the *field* value with

operand Argument for the operator

Swarm supports the following operators:

- `'@in'`: Match the field against any value in the operand
  - *Operand*: `[value1, value2, ...]`
- `'@nin'`: Match the field to be **not equal** to any value in the operand or to be absent from node
  - *Operand*: `[value1, value2, ...]`
- `'@like'`: Match a translated string against the value in the operand for given language
  - *Operand*: `{'language':'string value'}`
- `'@is'`: It's like operator "@like" but compares equality instead of matching.
  - *Operand*: `{'language':'string value'}`
- `'@regex'`: Match the field against regular expression specified in operand value.
  - *Operand*: `'regular_expression'`

Note: PERL compatible regular expression (PCRE) syntax is used and matching is performed from the beginning of field value.

- `'@gt'`: Match the field against any value greater than the operand value
  - *Operand*: `value`
- `'@lt'`: Match the field against any value lower than the operand value
  - *Operand*: `value`
- `'@gte'`: Match the field against any value equal to or greater than the operand value
  - *Operand*: `value`
- `'@lte'`: Match the field against any value equal to or lower than the operand value
  - *Operand*: `value`
- `'nested-field-name'`: Name of the nested field of complex node field to match against given value or expression
  - *Operand*: `value` or `expression`

Logical operations syntax:

```
{
    'level': {
        'operator': operand
    }
}
```

operator Operation to apply to specified filter in *operand* field

operand Corresponding filter expression

- `'@or'`: Perform logical OR comparison on supplied array of two or more expressions and match the nodes that satisfy at least one of the expressions
  - *Operand*: [{ <expression1> }, { <expression2> }, ...]

  example: { event: { @or: [{ price: { @lt: 20 } }, { price: 100 }] }
- `'@and'`: Perform logical AND comparison on supplied array of two or more expressions and match the nodes that satisfy all expressions
  - *Operand*: [{ <expression1> }, { <expression2> }, ...]

1.12.2 Subscribing and receiving updates

Subscribing to the query means that if any changes happen to the result of the query, those changes will automatically be pushed to your session. In case of Websocket, the changes will just arrive via the Websocket connection. In case of Long poll, a `whats_up()` request will fetch the changes.

Example request:

```
{
    "command": "whats_up"
}
```

If `subscribe` was set to `false`, or omitted, the initial `get()` query would simply return the dataset without further updates of changes to that particular query.

If `subscribe` was set to `true`, the query response would also contain subscription ID (`subid`) value along with dataset.

```
{
    ...
    "data": {
        "data": {
            "sport": {
                ...
            }
        },
        "subid": "5727868067050970589"
    }
}
```

Subscription ID is used for:

1. unsubscribing from specific query (`unsubscribe()`).
2. identifying specific query data in `whats_up()` response.

Example whats_up response which contains updates on several subscriptions:

```
{
    "code": 0,
    "rid": 0,
    "data": {
        "1593298234837013460": { // Subscription 1
                "sport": {
                    "844": { ... } // changed or new node
                }
        },
        "-86405724519794324": { // Subscription 2
                "region": {
                    "65537": null // removed node
                }
        },
        "5727868067050970589": { // Subscription 3
                "event": {
                    "923782145": {
                        "price": 5.35 // changed field
                    }
                }
        }
    }
}
```

Note: Update assumes data on added/changed/removed fields and nodes. Removed data will have value of `null` for field/node/level.

1.12.3 Unsubscribing from query updates

When query data updates are no longer needed, `unsubscribe()` command should be called specifying original query Subscription ID.

Example request:

```
{
    "command": "unsubscribe",
    "params": {
        "subid": "5727868067050970589"
    }
}
```

CHAPTER

TWO

CLIENT DEVELOPMENT - AUTHENTICATED OPERATIONS (PARTNERAPI VERSION)

This section of documentation is created for partners which have implemented PartnerAPI integration (according to "Partner API integration" documentation). User management (accounts and wallets) is done at partner side in this case, although SWARM provides unified API for placing bets regardless of that.

2.1 Logging in and out

So first of all before any user specific operation is done - user should be authenticated. In PartnerAPI case this is done by Partner at their website, but SWARM should be notified about that so session requested earlier with `request_session` command is marked as authenticated.

This is there `restore_login` command comes in help. Without it all further commands (for example placing bet) with fail with "Not authenticated" error.

Example request:

```
{
    "command": "restore_login",
    "params": {
        "user_id": 11111,
        "auth_token": "some-random-authToken-12345678"
    }
}
```

where:

user_id User ID in Partner system. Will be validated to match `user_id` in `GetAccountDetails` response.

auth_token Valid AuthToken for user.

Note: `restore_login` command will result in `GetAccountDetails` to OperatorBE and on successful call SWARM session will be marked as authenticated and will allow further operations.

Successful response example:

```
{
    "code": 0,
    "data": {
        "auth_token": "test_token_228",
        "user_id": 228
    }
}
```

As you can see `code` indicates success, and `user_id` and `auth_token` are echoed back.

Error response example:

```
{
    "code": 12,
    "data": "login error (1002)",
    "msg": "Invalid credentials"
}
```

`code` and `msg` here are self-descriptive. `data` contains error number forwarded from backend. See *Appendix D: Swarm Error Codes* for more information.

At some point it might be needed to `logout` current user.

Example of logout request:

```
{
    "command": "logout"
}
```

It doesn't receive any parameter and will just clear authentication information from current user's SWARM session.

2.2 Keeping user on-line

This part is very specific for PartnerAPI integration. Since AuthToken has an expiration time and when user is idle there should be some request which extends AuthToken lifetime. Otherwise PartnerAPI will consider user offline.

Usually RefreshToken or GetBalance request is made by client application every 30-60 seconds to keep user on-line.

To simulate this behavior in SWARM please issue following command periodically:

```
{
    "command": "get_balance"
}
```

2.3 Preparing and placing bets

Once user is authenticated he's eligible for placing bets. `event` node (with its `id` and `price` fields) is used as a betting unit (odd).

Placing bet is a process of composing betslip which includes:

- stake amount (entered by user)
- bet type (selected by user)
- single or multiple odds (selected by user) (consisting of event `id` and `price`)
- odd change mode (bet mode) (selected by user)

Once this information is ready its time to proceed to main betting process.

2.3.1 Placing bet

Placing bet is done by issuing `do_bet` command.

Example of do_bet request:

```
{
    "command": "do_bet",
    "params": {
        "type": 1,
        "mode": 0,
        "amount": 50.0,
        "bonus_id": 112233 //optional
        "bets":[
            {
                "event_id": 929407026,
                "price": 2.1
            }
            // ...
        ]
    }
}
```

where:

type Bet type (see *Appendix D: Swarm Error Codes* for list of possible values)

mode Mode - odd change mode (see *Appendix D: Swarm Error Codes* for list of possible values)

amount Stake amount

bonus_id Bonus id (in case when client want to place bonus bet) optional

bets List of bet event_ids and prices.

For single/ordinar stakes `bets` list will contain a single item. For other stakes - it is supposed to be a list of multiple items. For system stakes - additional `sys_bet` parameter is sent along with `do_bet` command.

```
{
    "command": "do_bet",
    "params": {
        "type": 3, // System bet
        "mode": 0,
        "amount": 50.0,
        "bets":[ // User selected 3 odds
            {
                "event_id": 929407026,
                "price": 2.1
            },
            {
                "event_id": 929407027,
                "price": 2.2
            },
            {
                "event_id": 929407028,
                "price": 2.3
            }
        ],
        "sys_bet": 2 // User expects 2 of 3 variants to win (2/3)
    }
}
```

where:

sys_bet Number of odds variants to win

Example of rejected bet response:

```
{
    "code": 0,
    "data": {
        "result": "2400",
        "details": [
            {
                "event_id": "929407026",
                "status": "OK",
                // some optional event details
                "price": 1.1, // latest event price
                "old_price": 1.2, // previous event price
                "new_event_id": 1234567, // in case event basis has changed - new event ID
                "new_basis": 3, // - and new basis
            }
        ]
    }
}
```

where:

result Reason (error code) of bet rejection (see *Appendix D: Swarm Error Codes* for list of values)

details Details on specific odd. For example if odd price has changed for specific event - it will contain `CHANGE_ODD` value. (see *Appendix D: Swarm Error Codes* for list of values)

Example of accepted bet response:

```
{
    "code": 0,
    "data": {
        "result": "OK",
        "details": {
            "bet_id": 12345678
        }
    }
}
```

where:

result (see *Appendix D: Swarm Error Codes* for list of successful values)

details shows resulting Bet ID.

Note: For placing multiple ordinar bets - each `do_bet` request should be sent independently.

Note: Bet outcomes will be reported back to PartnerAPI OperatorBE via MoneyDeposits calls.

2.3.2 Booking bet

Booking bet is done by issuing `book_bet` command.

Example of book_bet request:

```
{
    "command": "book_bet",
    "params": {
        "type": 1,
        "source": "1",
```

```
        "amount": null,
        "bets": [
            {
                "event_id": 197327425,
                "amount": 0.1
            }
            //...
        ]
    }
}
```

where:

type Bet type (see *Appendix D: Swarm Error Codes* for list of possible values)

source source, integer number to distinguish origin optional

amount Stake amount

bonus_id Bonus id (in case when client want to place bonus bet) optional

bets List of bet event_ids and prices.

For single/ordinar stakes `bets` list will contain a single item. For other stakes - it is supposed to be a list of multiple items. For system stakes - additional `sys_bet` parameter is sent along with `book_bet` command.

```
{
    "command": "book_bet",
    "params": {
        "type": 3,
        "source": "1",
        "amount": 0.1,
        "bets": [
            {
                "event_id": 197327397
            },
            {
                "event_id": 197470026
            }
        ],
        "sys_bet": 2
    }
}
```

where:

sys_bet Number of odds variants to win

Example of accepted bet response:

```
{
    "code": 0,
    "data": {
        "result": 0,
        "result_text": null,
        "details": {
            "number": 977586
        }
    }
}
```

where:

result (see *Appendix D: Swarm Error Codes* for list of successful values)

details shows resulting Bet ID.

Note: For placing multiple ordinar bets - each `book_bet` request should be sent independently.

Note: Bet outcomes will be reported back to PartnerAPI OperatorBE via MoneyDeposits calls.

2.3.3 Getting limits

In some situation before placing bet it is necessary to get and display maximum bet/win limit set on specific odd.

Example of get_max_bet request:

```
{
    "command": "get_max_bet",
    "params": {
        "events":[
            929343821,
            // 929343820,
            // 929314233
        ]
    }
}
```

where:

events List of events to return limit for. For an ordinal bet - its a single event list. For other bet types (for example express bets) - its a list of events and the return value is limit for total stake amount.

Example of get_max_bet response:

```
{
    "code": 0,
    "data": {
        "result": "7",
        "details": null
    },
}
```

where:

result Limit value. In value is a negative number - its an error number.

details Optional description in case of error.

Note: `get_max_bet` considers the following limits and returns minimal value of:

- limit set for user (if any)
- limit set on specific game (if any)
- global partner limit for live/prematch games

2.3.4 Bet History

To fetch and display user's bet history issue `bet_history` command. It will return list of user's bets along with outcome/original event information.

Example of bet_history request:

```
{
    "command":"bet_history",
    "params":{
        "where":{
            // use case 1
            "from_date":1401566400,
            "to_date":1404158400,
            "outcome":0,
            "bet_type":1
            // use case 2
            // "bet_id": 910891367
        }
    }
}
```

where:

where optional list of filters to apply. if left empty - it will return user's recent bets without any filtering.

- `from_date` and `to_date` - UTC timestamps, allowing to filter bet history by date range
- `outcome` - filters by bet outcome (see *Appendix D: Swarm Error Codes* for list of values)
- `bet_type` - filters by bet type
- `bet_id` - return specified bet's information only

Example of bet_history response:

```
{
    "code":0,
    "data":{
        "bets":[
            // List of bets
            {
                "id":"910891367", // Bet ID
                "date_time":"1404296442",  // Bet date (UTC timestamp)
                "type":"1", // Bet Type
                "amount":"0.2", // Bet amount
                "currency":"USD", // Bet currency
                "k":"1.23", // Bet price/coefficient
                "outcome":"3", // Bet outcome
                "payout":"0.25", // Payout amount
                "events":[ // Additional information about event user has placed on
                "cash_out":45.0, // CashOut amount by which client can cashOut the bet (optional)
                    {
                        "id":"910860797",
                        "sport_id":"848"
                        "sport_name":"Tennis",
                        "competition_name":"Italy - ATP Challenger Todi - Clay (LIVE)",
                        "game_id":"910636857",
                        "game_name":"Taro Daniel - Flavio Cipolla",
                        "game_info":"0 : 0",
                        "game_information":"<GameInfo passTeam=\"team2\" currentGameState=\"finished\
                        "game_start_date":"2014-07-02 09:30:00",
                        "team1":"Taro Daniel",
                        "team2":"Flavio Cipolla",
                        "is_live":"1",
                        "basis":"",
```

```
                        "event_name":"W1",
                        "event_info":"0 : 0, (4:1) 15:15*; ",
                        "name_first_number":"",
                        "name_last_number":"",
                        "outcome":"3",
                        "coeficient":"1.23",
                    }
                ]
            },
            // ...
        ]
    }
}
```

CHAPTER

THREE

DATA

3.1 Hierarchy

Swarm data is hierarchic in nature. A single unit of data is a *node* organized into 2 top-level collections (*sources*), and various 2nd-tier *levels*:

- **betting: All betting-related data**
  - sport
  - region
  - competition
  - game
  - market
  - event
- **user: All user-related data**
  - profile
- **messages: All user-messages related data**
  - messages

All names in the list above are self-explanatory enough to make description redundant.

Each of the levels has its own set of attributes shared by all nodes under that level. There are also two attributes which are shared across all levels and sources: **id** and **order**. First represents the ID of the node, and is unique across the entire data set. Second represents the order of the node in a list for the purpose of sorting. Every node has an `id`, but not all nodes have `order`.

3.2 Localization

Texts in Swarm are returned in the language specified in `request_session` command.

3.3 Levels

3.3.1 sport

name **localized** Name of the sport

alias Alias

order Order

3.3.2 region

name [localized] Name of the region

alias Alias

order Order

3.3.3 competition

name [localized] Name of the competition

order [optional] Order

3.3.4 game

type Type of the game. Possible values:

0: Prematch **1:** Live **2:** Future Live

start_ts Timestamp of the start of the game

is_started Whether or not the game is started

is_blocked Whether or not the game is blocked

events_count Number of events in the game

team1_name [localized] Name of the first team playing in the game

team2_name [localized optional] Name of the second team playing in the game

team1_id Unique ID of team1

team2_id Unique ID of team2

game_number Secondary alias ID of the game for aesthetic purposes

text_info [optional live] Text-only single line game info line

info [optional live object] Additional information about the game

- ***current_game_state*** Current state of the game
- ***current_game_time*** Current time of the game
- ***field*** Type of the game field
- ***shirt1_color*** RGB color of the shirt of the first team
- ***short1_color*** RGB color of the shorts of the first team
- ***shirt2_color*** RGB color of the shirt of the second team
- ***short2_color*** RGB color of the shorts of the second team
- ***score1*** Score of the first team
- ***score2*** Score of the second team

add_info Additional information about the game

stats optional live map Additional statistics about the game

***key*:** Type of stat

***value*:** object

***team1*:** Value of the stat for team 1

***team2*:** Value of the stat for team 2

add_info Extra information about the stat

live_events optional live array In-game chronological events for live games

event_type Type of event

team Team the event applies to. Possible values:

team1: Team 1 **team2:** Team 2

add_info Extra information about the event, for example time of the event

is_feed_available if scout information feed is available (see *last_event*)

last_event optional live Last in-game event info (as reported by scouts)

type type of event. see *Appendix C: In-Game Events* for list of values

more fields here which are mostly self-descriptive

express_min_len optional if value is 1 - no bet chaining is allowed for this game

exclude_ids optional list of games which cannot be chained with this game bets

tv_type optional if game video stream is available - ID of video provider

video_id optional if game video stream is available - video stream ID

video_id2 optional if game video stream is available - alternative video stream ID

top_game optional if game is marked as "popular"

descr optional **:sup:** ***prematch*** Additional game description

comp_country team1country team2country

Competition/Team1/Team2 country

flive if corresponding game is already created in Future Live (upcoming) list

visible_in_prematch optional live is live game has to be shown in prematch also

3.3.5 market

name localized Name of the market

type optional Type of the market

order Order

base optional Base of the market

col_count Number of columns to represent this market

express_id optional Grouping for restriction of express betting. Markets with the same `express_id` cannot be chained together

group_id optional prematch ID of the market group

group_name localized optional prematch Name of the market group

cashout optional Event sets for this market support cashOut

3.3.6 event

***name*:** localized Name of the event

order Order

type optional Type of the event

price Price

base optional Base of the event

sp_enabled optional If start price (SP) is enabled for event

ew_allowed optional if EachWay betting is enabled for event

3.3.7 profile

name Name of the user

super_bet Super bet

last_name Last name of the user

casino_promo Casino promo

address Address of the user

currency_name Three-letter name of the currency

credit_renew_time Timestamp of the credit renewal

first_name First name of the user

bonus_id Bonus ID

reg_date Registration date of the user

points_balance Points balance

last_read_message optional Date of the last read message

email Email address of the user

username Username

doc_number optional Doc number

reg_info_incomplete Is registration information incomplete

exclude_date optional Exclude date

currency_rate Currency rate

currency_id Currency ID

initial_balance Initial balance

gender Single-letter gender of the user

unread_count Count of unread messages

games Games

birth_date Birth date of the user

balance Balance

unique_id Seven-letter unique ID of the user

has_free_bets Whether free bets are enabled for user or not.

3.3.8 messages

subject Message subject

body Message body

CHAPTER

FOUR

APPENDIX A: SWARM ERROR CODES

Code	Description
0	No error
1	Bad Request
2	Invalid Command
3	Service Unavailable
4	Request Timeout
5	No Session
6	Subscription not found
7	Not subscribed
10	Invalid Level
11	Invalid Field
12	Invalid Credentials
20	Not enough balance for operation
21	Operation not allowed
22	Limit reached
23	Service temporary is down
99	Unknown Error

CHAPTER

FIVE

APPENDIX B: LANGUAGE CODES

Code	Language
eng	English
fra	French
spa	Spanish
por_2	Portuguese *
por	Portuguese (Brazil) *
ita	Italian
ger	German
dut	Dutch *
grk	Greek *
rou	Romanian *
srp	Serbian *
rus	Russian
ukr	Ukranian *
est	Estonian *
lav	Latvian *
lit	Lithuanian
arm	Armenian
geo	Georgian
arb	Arabic
far	Persian (Farsi)
tur	Turkish
zho	Chinese *
kor	Korean *

If language code is not supported, it will fallback to English.

ask BetConstruct for more information.

CHAPTER

SIX

APPENDIX C: IN-GAME EVENTS

6.1 Football/Soccer

- Goal
- RedCard
- YellowCard
- Corner
- Penalty
- Substitution
- BallSafe
- DangerousAttack
- KickOff
- GoalKick
- FreeKick
- ThrowIn
- ShotOffTarget
- ShotOnTarget
- Offside
- GoalkeeperSave
- ShotBlocked
- NotStarted
- FirstHalf
- HalfTime
- SecondHalf
- PreExtraHalf
- ExtraTimeFirstHalf
- ExtraTimeHalfTime
- ExtraTimeSecondHalf
- Finished

- Attack

6.2 Tennis

- FirstSet
- SecondSet
- ThirdSet
- FourthSet
- FifthSet
- Point
- BallInPlay
- ServiceFault
- DoubleFault
- Ace
- InjuryBreak
- RainDelay
- Timeout
- Finished

Note: more sports event information to come.

CHAPTER

SEVEN

APPENDIX D: SWARM ERROR CODES

7.1 Bet Type values and meaning

Bet type is used to specify what kind of stakes are done by user:

1	Single/Ordinar
2	Express/Parlay
3	System
4	Chain

7.2 Bet Mode values and meaning

Bet Mode is used to change betting mode for the client and could be one of the following:

-1	Super bet mode
0	Accept bet ONLY if odd has not been changed
1	Accept bet if odd has not been changed OR if odd has been increased
2	Accept bet with ANY odd changes

7.3 Bet `result` values

Code	Description
OK	Accepted
ONHOLD	Accepted, but placed on hold for review
500-999	Forwarded error code from Operator's PartnerAPI backend
1000	Internal Error
1001	Null value for argument
1002	Wrong Login/Password
1003	User blocked
1004	User dismissed
1005	Password error
1008	Logging in the page is not possible, since user is not activated or verified
1009	Such a verification code does not exist.
1012	Incorrect phone number
1013	Password is too short
1014	Failed to send verification SMS

Continued on next page

Table 7.1 – continued from previous page

Code	Description
1023	User is not verified via email
1099	Fork exception
1100	Game is already started
1102	Game start time is already past
1103	Bet editing time is already past
1104	Bet is payed
1105	Bet status not fixed
1106	Bet lose
1107	Bet is online
1108	Wrong value for coefficient
1109	Wrong value for amount (in case of system bet - amount is less than minimum allowed)
1021	Pass should contain at least 8 chars: upper and lowercase English letters, at least one digit and no space
1112	Request is already paid!
1113	Request is already stored!
1117	Wrong login or E-mail
1118	Duplicate Login
1119	Duplicate EMail
1120	Duplicate nickname
1122	Duplicate personal Id
1123	Duplicate doc number
1124	Amount is not in valid range
1125	Bet type error
1126	Bet declined by SKKS
1127	Duplicate phone number
1150	You yet are not allowed to bet on the given event yet
1151	Duplicate Facebook ID
1170	Card lot blocked
1171	Scratch card already activated
1172	Scratch card blocked
1174	Wrong scratch card currency (not supported for user currency)
1200	Wrong value exception
1273	Wrong scratch card number
1300	Double value exception
1400	Double event exception
1500	Limit exception
1550	The sum exceeds maximum allowable limit
1560	The sum is less than minimum allowable limit
1600	There is going the correction of coefficient.
1700	Wrong access exception
1800	Odds is changed from %s to %s
1900	The events can be included only in the express
2000	Odds restriction exception
2005	Cashdesk not found
2006	Cashdesk not registered
2007	Currency mismatch
2008	Client excluded
2009	Client locked
2018	Email should net be an empty
2020	First name should not be an empty

Continued on next page

Table 7.1 – continued from previous page

Code	Description
2024	Invalid email
2033	Market suspended
2036	Game suspended
2048	Wrong region
2051	Partner api access not activated
2052	Partner api Client Balance Error
2053	Partner api client limit error
2054	Partner api empty method
2055	Partner api empty request body
2056	Partner api max allowable limit
2057	Partner api min allowable limit
2058	Partner api PassToken error
2059	Partner api timestamp expired
2060	Partner api token expired
2061	Partner api user blocked
2062	Partner api wrong hash
2063	Partner api wrong login email
2064	Partner api wrong access
2065	Partner not found
2066	Partner commercial fee not found
2072	Reset code expired
2074	Same password and login
2075	Selection not found
2076	Selection count mismatch
2077	Event suspended
2078	Sport mismatch
2080	Sport not supported for the partner
2100	Payment restriction exception
2200	Client limit exception
2302	Terminal balance exception
2403	There are active requests for this client
2405	Client bonus not found
2406	Partner bonus not found
2407	Client has active bonus
2408	Invalid client verification step
2409	Partner setting not allow this type of self exclusion
2410	Invalid self exclusion type
2411	Invalid client limit type
2412	Invalid client bonus
2413	Client restricted for action
2414	The events can be included only in the ordinar
2415	Partner not supported test client
2416	Partner not using loyalty program
2417	Point exchange range exceed
2418	Client not using loyalty program
2419	Office limit exception
2420	Client has accepted bonus
2421	Partner api error
2422	Team not found

Continued on next page

Table 7.1 – continued from previous page

Code	Description
2423	Invalid client verification step state
2424	Partner sportsbook currency setting error
2425	Client bet stake min limit error
2091	Password expired
2093	Username already exist
2098	Wrong currency code
3001	Currency not supported
3015	Negative amount
3019	Bet selections cannot be chained together
2403	Pending withdrawal requests
2404	Cash out not allowed
2405	Bonus not found
2406	Partner bonus not found
2407	Client has active bonus
2408	Invalid client verification step
2409	Partner setting not allow this type of self exclusion
2410	Invalid self exclusion type
2411	Invalid client limit type
2412	Invalid client bonus
2413	Client restricted for action
2414	Selection singles only
2415	Partner not supported test client
2416	Partner not using loyalty program
2417	Point exchange range exceed
2418	Client not using loyalty program
2419	Partner limit amount exceed
2420	Client has accepted bonus
2421	Partner api error
2422	Team not found
2423	Invalid client verification step state
2424	Partner sports book currency setting
2425	Client bet min stake limit error
2426	Max deposit request sum
2427	Email wrong hash
2428	Client already self excluded
2429	Transaction amount exceeds frozen money
2430	Wrong hash
2431	PartnerMismatch
2432	MatchNotVisible
2433	LoyaltyLevelNotFount

7.4 Balance History `operation` values

Code	Description
0	New Bets
1	Winning Bets
2	Returned Bet
3	Deposit
4	Card Deposit
5	Bonus
6	Bonus Bet
7	Commission
8	Withdrawal
9	Correction Up
302	Correction Down
10	Deposit by payment system
12	Withdrawal request
13	Authorized Withdrawal
14	Withdrawal denied
15	Withdrawal paid
23	In the process of revision
24	Removed for recalculation
29	Free Bet Bonus received
30	Wagering Bonus received
31	Transfer from Gaming Wallet
32	Transfer to Gaming Wallet
37	Declined Superbet
39	Bet on hold
40	Bet cashout

7.5 Bet details' `status` values

Code	Description
OK	No problems on specific odd
CHANGE_ODD	Odd value (price) has been changed
EVENT_LOCKED	Event is locked
LIMIT_ERROR	Event limit exceeded
BASIS_CHANGED	Event basis has changed
GAME_STARTED	Game is already started

7.6 Bet outcomes

Value	Bet Outcome
-2	WaitingPartner
-1	Rejected
0	Waiting/not calculated
1	Lost
2	Returned (void)
3	Won
5	CashedOut

CHAPTER

EIGHT

COPYRIGHT