**Bantam project protocol description**

Version 0.0, draft, 05, May, 2018

1. **Introduction**

Bantam protocol describes the communication between client and server software. Server software is running somewhere on the Subscale’s servers, while client can be written with any language. The communication is based on WebSocket protocol.

All the messages within communication are sent as JSON UTF8 messages. Max message size is 256kb. Each client connection has several limits controlled by server side that are used to avoid server overloading.

The communication is done with this steps:

1. Client connects to the server with URL: ws://api.bantam.net
2. Server will send “hello” message describing the connection details
3. Client should send “hello” command to the server
4. Every 10 seconds, server sends “ping” request, the client should answer with “pong”. If the client fails to answer with valid response for more than 10 seconds, the server will close connection.
5. At this stage, we can say, that client and server have live data channel that can be used to access data and send additional commands.
6. **General commands structure**

There are three types of messages: requests, response, data, each of the has general structure

{“type”: “message type”, “opaque”: integer\_identifier, other fields…}

Where type can be one of valid message types, described below.

“Opaque” field is used to identify message sequence. Each response should have the same opaque identifier and the request. Data messages may not contain “opaque” field.

1. **Detailed commands description**
   1. **HELLO**

Afther client is connected, the server will send this request:

{“type”:”hello”, “server\_info”: “Server information”, “protocol\_version”: “1.0”, “compression”: “false”, “opaque”: 1, “authentication”: “none”}

Client should response:

{“type”:”hello”, “opaque”:1, “protocol\_version”: “1.0”}

* 1. **PING**

Every 10 seconds, server sends

{“type”:”ping”, “opaque”: 1}

Client responds

{“type”:”pong”, “opaque”: 1}

* 1. **SUBSCRIBE**

This command is used by client to subscribe into some real-time data stream.

{“type”: “subscribe”, “opaque”: 1, “channel”: “channel name”, “options”: {channel options…}}

Where channel name is some unique data channel name. All possible channels can be listed with “GET” command. Options field is optional and can be used to give additional details to the data channel

Server will response

{“type”: “subscribed”, “opaque”: 1, “channel”: “channel name”}

* 1. **UNSUBSCRIBE**

This command is used to unsubscribe from a channel, that client was subscribed earlier

Client’s request:

{“type”: “unsubscribe”, “opaque”: 1, “channel”: “channel name”}

Server’s response:

{“type”: “unsubscribed”, “opaque”: 1, “channel”: “channel name”}

* 1. **GET**

This command can be used to get some data resource from server. Typical command structure:

{“type”: “get”, “opaque”: 1, “resource”: “resource-id”, optional fields…}

Where resource-id is used to identify the resource that should be read.

Server will respond with:

{“type”: “get”, “opaque”: 1, “resource”: “resource-id”, “content”: resource\_content}

1. **Resource list**
   1. **Channel list: “channels”**

This resource contains the list of all possible channels available for subscription

1. **Live data channels**

After client is connected to the channel, the server will start sending the channel’s data with “data” messages.

* 1. **Order book channels**

Order book channels are used to get real-time order books from Bantam network server. Each order book channel can have two types of data: “snapshot” and “update”, the general data message structure is described below:

{“type”: “data”, “opaque”: 1, “timestamp”: milliseconds, “content\_type”: “snapshot/update”, “data”: {“bids”:[[price, volume], [price, volume],…], “asks”: [[price, volume], [price, volume],…]}}

Where timestamp contains server time in milliseconds since epoch (Unix time) UTC+0 (time is always UTC+0 time zone).

When receiving the “update” message, the client’s recommended algorithm is:

For each bid, ask, find the local stored order book data of same price. If update price is zero (0), then remove this item from order book, otherwise replace the volume at local order book with new data.

1. **Error codes**

If something went wrong, the error message will be generated as the response to the client’s request:

{“type”: “error”, “opaque”: integer\_identifier, “description”: “some text description”, “code”: “error\_type”}

Opaque field can be used to identify request the failed request