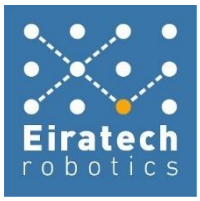



Eiratech robotics limited

# Websocket JSON Sky data exchange protocol.

Author: Alexey Malyshev

## Eiratech Robotics Confidential



## Revision History

| Revision | ECO # | Document Status | Date | Author | Description of change |
|---|---|---|---|---|---|
| E00 | 00?? | Draft | 28/09/2021 | Alexey Malyshev | Initial Draft for review |
| | | | | | |
| | | | | | |

Contents

# Introduction

The Sky communication protocol is intended for use as a highly reliable host-to-host protocol between hosts in packet-switched computer communication networks, and in interconnected systems of such networks.

This document describes the functions to be performed by the Sky communication protocol, the program that implements it, and its interface to programs or users that require its services.

# Protocol layering

| |
|---|
| Application specific protocol |
| Sky proto |
| Websocket |
| TCP |
| Internet protocol |
| Communication network |

Sky communication protocol is a base protocol for application specific protocols.

Since websocket protocol is message oriented not necessary to control message begin/end tokens, message boundaries aligned by the protocol. Skyproto top level JSON messages incapsulated to websocket messages

# Message structure

Messages consist of the set of nested JSON objects:

```
{
	“Top level wrapper” :
	{
		“Payload”:
		{
		}
	}
}
```

## Top level wrapper

Top level wrapper defines message payload. The message payload is a nested JSON object.

```
{
	"type": "msg_type",
	"msg_type":
	{
	}
}
```

Available next message types:

1. handshake_request – client sends to server to initialize new connection or reconnect after disconnection
2. handshake_response – server response on a client connection request
3. msg – generic message for both sides

# Connect

## Websocket layer.

Server opens listening socket to process incoming connection requests from the clients. Client creates socket and sends connection requests to the server. Client responsible for keeping connection alive and reconnection. Server keeps connection 24 hours after last received message from the client.

Server generates unique connection id for each new connection.

Client sends heartbeat messages 10 seconds after last sent message to keep connection alive and test end to end communication.

## Handshake.

### Messages format

#### *Handshake request.*

A client sends handshake request message to the server to initialize connection. The message has next structure:

```
{
  "client_id": "robot 1",
  "connection_id": "00112233-4455-6677-8899-aabbccddeeff",
  "last_msg_id": 12345
}
```

Client_id – string to identify client name. Used to recognize client on different services and on reconnection.

Connection_id – RFC 4122 UUID string to identify connection.

last_msg_id – number of messages received from the server.

#### *Handshake response.*

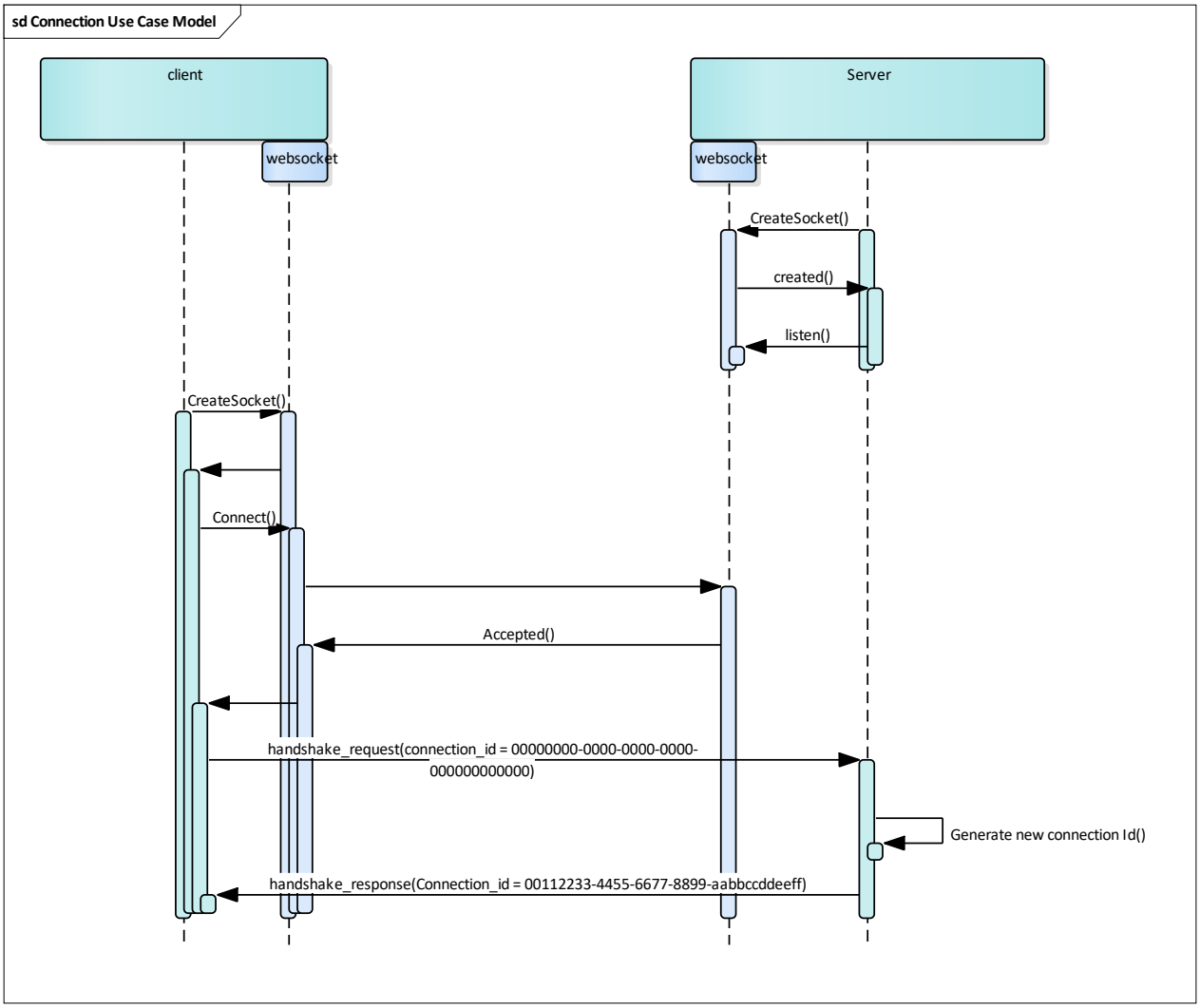
A server sends handshake response to reply to a client on a handshake request. The message has next structure:

```
{
  "connection_id": "00112233-4455-6677-8899-aabbccddeeff",
  "last_msg_id": 12345
}
```

Connection_id – RFC 4122 UUID string to identify connection.

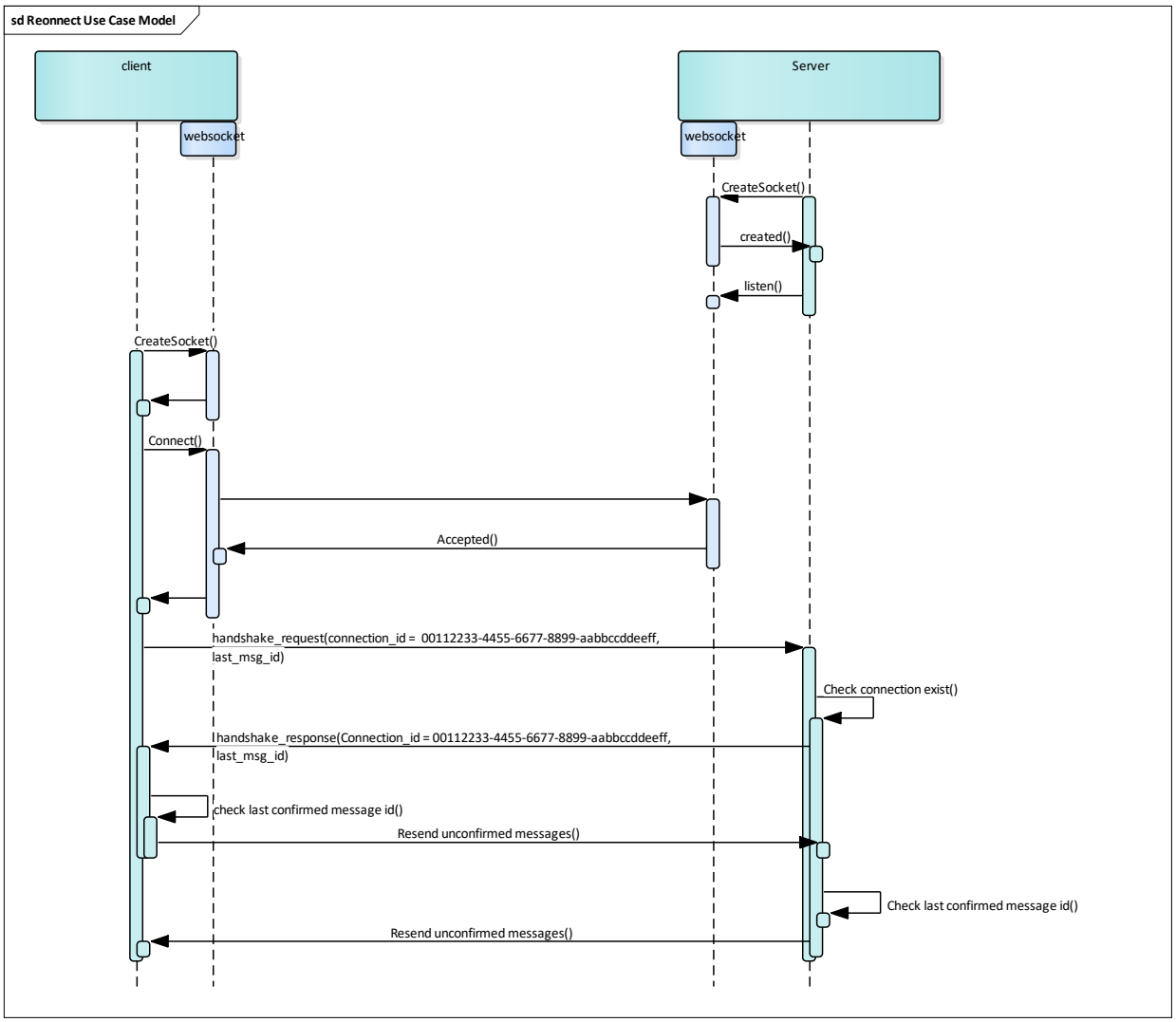
last_msg_id – number of messages received from the client identified by connection_id. The message counter has to be reset if the server generates new connection_id.

## New connection



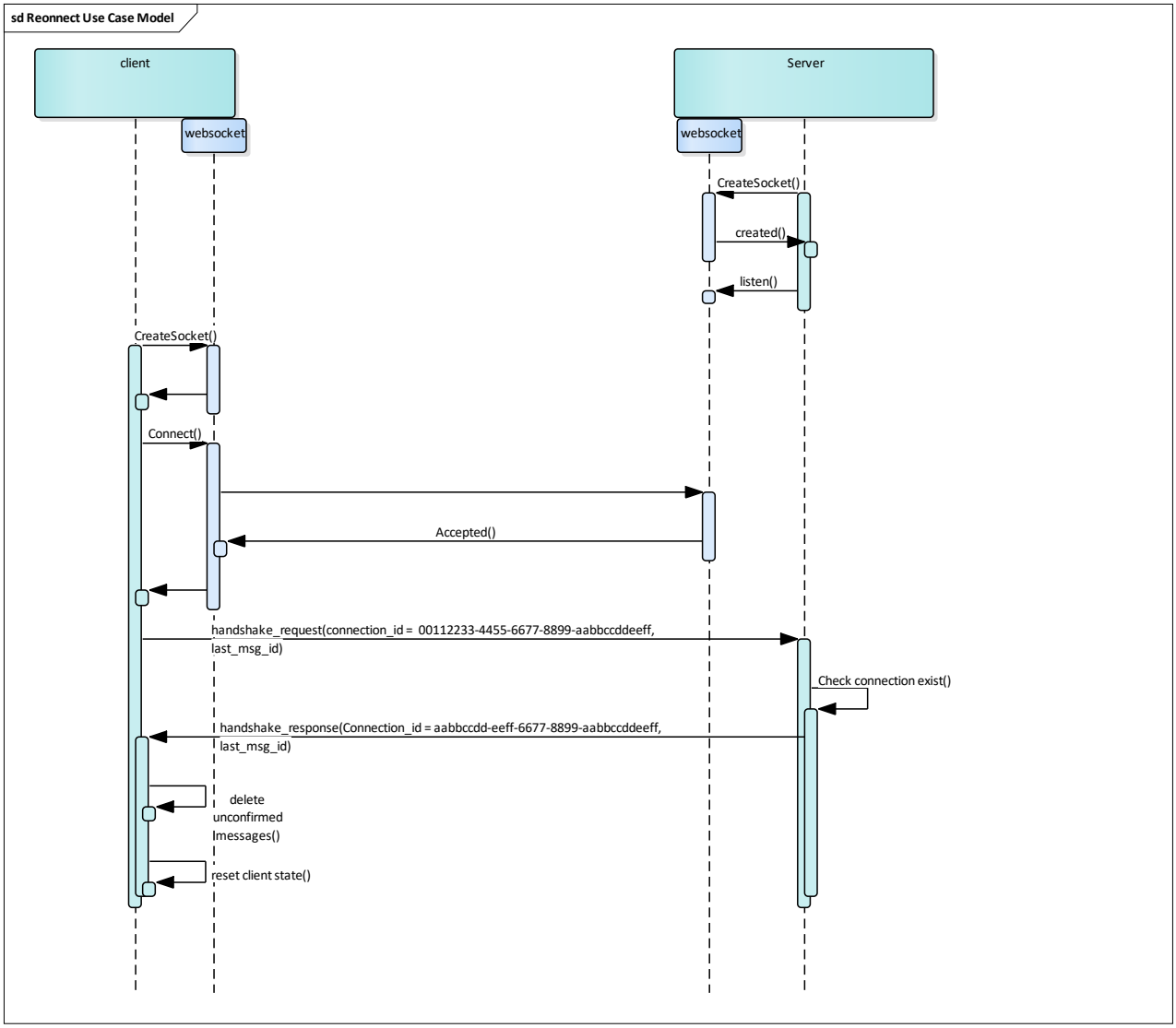

On first connection client sends nil UUID “00000000-0000-0000-0000-000000000000” to inform server o consider connection as new. The server generates new UUID and sends it to the client. The client stores received connection_id to use it if connection will be dropped. We need it to restore system state if connection was dropped but nor client nor server was restarted. On new connection last_message_id is always zero.

## Reconnect without server restart.




On reconnection client sends last UUID received from the server. The server checks connection_id to find connection in a connection list. If the UUID found, the server responds with same UUID and number of received messages. Client/server continue communication.

## Reconnect after server restart




On reconnection client sends last UUID received from the server. Server check connection_id to find connection in a connection list. If the server can't find connection_id in the local connection list, the server generates new UUID and sends it to the client with zero in last_message_id field. The client has to run connection restart procedure.

## Generic messages send/receive

Wrapper for generic message has next format:

```
{
  "type": "msg",
  "msg": {
    "id": 1234,
    "payload": "payload_type_a",
    "payload_type_a": {}
  }
}
```

Id – number of received messages. Since we use tcp/websockets, the order of messages is guaranteed, no reasons to send message id in each message. We can count received messages on client side and send number of received messages. Also, not necessary to confirm each message, we can send number of received messages with a message (regular or keep alive). Otherwise, synchronization will be done on reconnection.

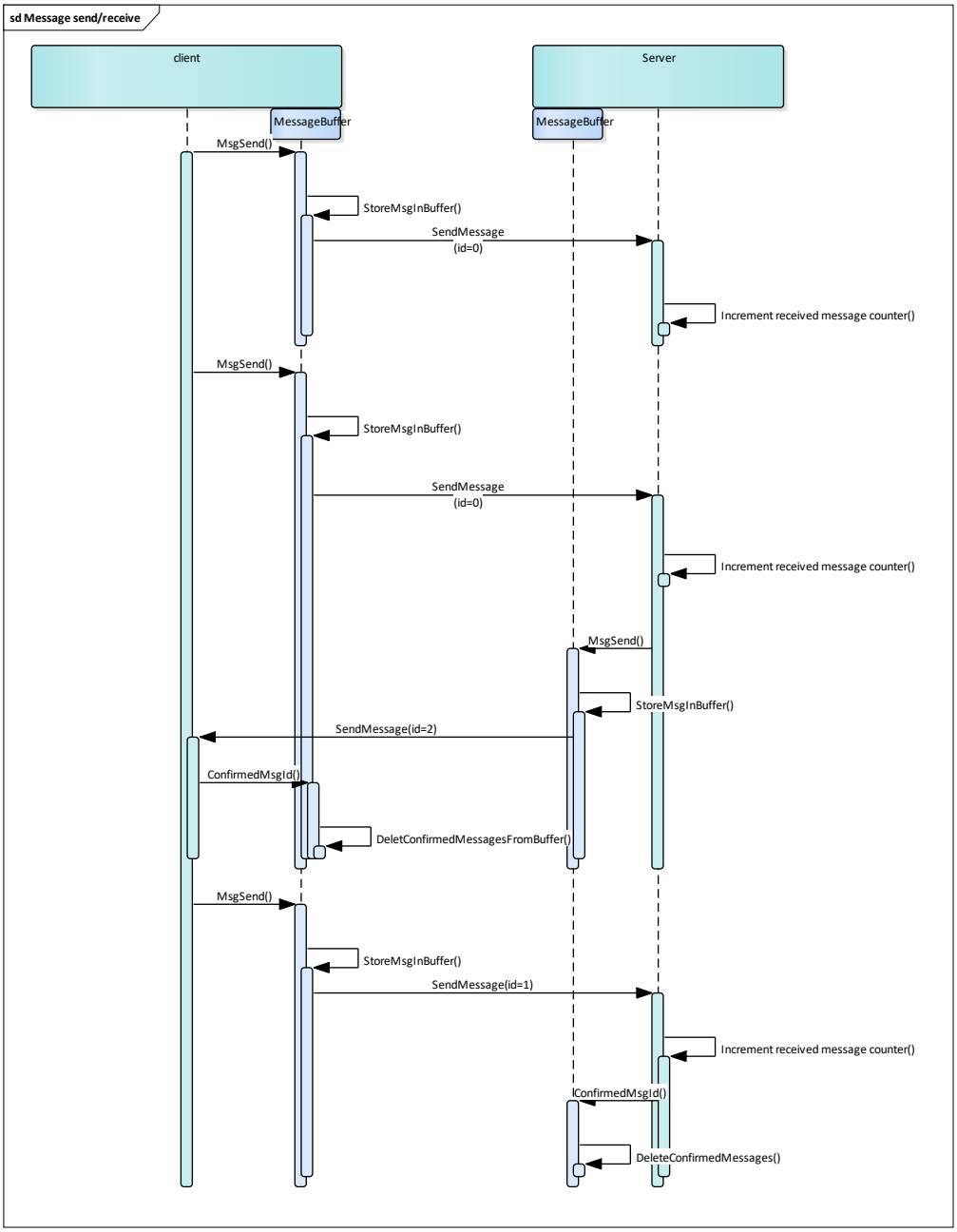
Payload – string field specify the type of the incapsulated message object. List of possible objects specified in a protocol extension document.

Sender indexing each message and stores it in a buffer. Receiver count received messages. The sender add number of received messages to the id field of the packet. On received message the node check the id field and delete all messages with index les or equal to received id. On reconnection the node receives last_msg_id received by other side and retransmit all unreceived messages.

Node A <-> Node B
node A sent 5 messages, node B received 3
node B sent 10 messages, node A received 9
after reconnection
node A has last_message_id = 9
node B has last_message_id = 3

node A sends a message, node B receives message, increments received message counter
node B sends a message puts message counter value to id field
node A receives a message, check id field, remove all messages up to id in the buffer

node A sent message 1, id field = 0, node B got message, inc counter to 1
node A sent message 2, id field = 0, node B got message, inc counter to 2
node A sent message 3, id field = 0
node A has messages 1, 2, 3 in a buffer
node B sent message 1, id field = 2 , node A got message, inc counter to 1, check Id, removes messages 1 and 2 from the buffer
node B got message, inc counter to 3


sd Message send/receive
client
Server
MessageBuffer
MessageBuffer
MsgSend()
StoreMsgInBuffer()
SendMessage
(id=0)
Increment received message counter()
MsgSend()
StoreMsgInBuffer()
SendMessage
(id=0)
Increment received message counter()
MsgSend()
StoreMsgInBuffer()
SendMessage(id=2)
ConfirmedMsgId()
DeletConfirmedMessagesFromBuffer()
MsgSend()
StoreMsgInBuffer()
SendMessage(id=1)
Increment received message counter()
ConfirmedMsgId()
DeleteConfirmedMessages()

## Appendix A. message schema

```
{
  "$id": "http://example.com/example.json",
  "$schema": "http://json-schema.org/draft-07/schema",
  "default": {},
  "description": "The root schema comprises the top level message
wrapper JSON document.",
  "examples": [
    {
      "type": "msg",
      "msg": {
        "id": 1234,
        "payload": "payload_type_a",
        "payload_type_a": {}
      }
    }
  ],
  "required": [
    "type"
  ],
  "title": "top level message schema",
  "type": "object",
  "properties": {
    "type": {
      "$id": "#/properties/type",
      "default": "",
      "description": "specify payload message type",
      "enum": [
        "msg"
      ],
      "examples": [
        "msg"
      ],
      "title": "message type",
      "type": "string"
    },
    "msg": {
      "$id": "#/properties/msg",
      "default": {},
      "description": "simple message payload schema.",
      "examples": [
        {
          "id": 1234,
          "payload": "payload_type_a",
          "payload_type_a": {}
        }
      ],
      "required": [
        "id",
        "payload"
      ],
      "title": "The msg schema",
      "type": "object",
      "properties": {
        "id": {
          "$id": "#/properties/msg/properties/id",
          "default": 0,
```

```
          "description": "Number of received messages",
          "examples": [
            1234
          ],
          "title": "The id schema",
          "minimum": 0,
          "type": "integer"
        },
        "payload": {
          "$id": "#/properties/msg/properties/payload",
          "default": "",
          "description": "message payload type",
          "examples": [
            "payload_type_a",
            "payload_type_b"
          ],
          "title": "The payload schema",
          "enum": [
            "payload_type_a",
            "payload_type_b"
          ],
          "type": "string"
        }
      },
      "additionalProperties": true
    }
  },
  "additionalProperties": false
}
```

## Appendix B. handshake request schema.

```
{
  "$id": "http://example.com/example.json",
  "$schema": "http://json-schema.org/draft-07/schema",
  "default": {},
  "required": [
    "handshake_request"
  ],
  "type": "object",
  "properties": {
    "handshake_request": {
      "$id": "#/properties/handshake_request",
      "type": "object",
      "title": "The handshake schema",
      "description": "handshake message ",
      "default": {},
      "examples": [
        {
          "client_id": "robot 1",
          "connection_id": "00112233-4455-6677-8899-aabbccddeeff",
          "last_msg_id": 12345
        }
      ],
      "required": [
        "client_id",
        "connection_id",
        "last_msg_id"
      ],
      "properties": {
        "client_id": {
          "$id": "#/properties/handshake_request/client_id",
          "type": "string",
          "title": "The client_id schema",
          "description": "client id",
          "default": {},
          "examples": [
            "robot 1",
            "taskhub"
          ]
        },
        "connection_id": {
          "$id": "#/properties/handshake_request/connection_id",
          "type": "string",
          "title": "The connection_id schema",
          "description": "client connection id",
          "default": {},
          "examples": [
            "00112233-4455-6677-8899-aabbccddeeff",
            "00000000-0000-0000-0000-000000000000"
          ]
        },
        "last_msg_id": {
          "$id": "#/properties/handshake_request/last_msg_id",
          "type": "number",
          "title": "The last message id schema",
          "description": "last received message id before
disconnection",
```

```
          "default": {},
          "minValue": 0,
          "examples": [
            "00112233-4455-6677-8899-aabbccddeeff"
          ]
        }
      },
      "additionalProperties": false
    },
    "additionalProperties": false
  },
  "additionalProperties": false
}
```

## Appendix C. handshake response schema

```
{
  "$id": "http://example.com/example.json",
  "$schema": "http://json-schema.org/draft-07/schema",
  "default": {},
  "required": [
    "handshake_response"
  ],
  "type": "object",
  "properties": {
    "handshake_response": {
      "$id": "#/properties/handshake_response",
      "type": "object",
      "title": "The handshake schema",
      "description": "handshake message ",
      "default": {},
      "examples": [
        {
          "connection_id": "00112233-4455-6677-8899-aabbccddeeff",
          "last_msg_id": 12345
        }
      ],
      "required": [
        "connection_id",
        "last_msg_id"
      ],
      "properties": {
        "connection_id": {
          "$id": "#/properties/handshake_response/connection_id",
          "type": "string",
          "title": "The connection_id schema",
          "description": "client connection id",
          "default": {},
          "examples": [
            "00112233-4455-6677-8899-aabbccddeeff"
          ]
        },
        "last_msg_id": {
          "$id": "#/properties/handshake_response/last_msg_id",
          "type": "number",
          "title": "The last message id schema",
          "description": "last received message id before
disconnection",
          "default": {},
          "minValue": 0,
          "examples": [
            "00112233-4455-6677-8899-aabbccddeeff"
          ]
        }
      },
      "additionalProperties": false
    },
    "additionalProperties": false
  },
  "additionalProperties": false
}
```