LAPSNAPPER APPLICATION PROGRAMMING INTERFACE SPECIFICATION v.5.20

Last updated December 17, 2025

Copyright © 2026 LapSnapper

Website address: www.lapsnapper.com

The information in this document is subject to change without prior notice. **LapSnapper** reserves the right to modify or enhance its products and content, without the obligation to notify any individual or organization of such changes or improvements. For the latest updates and additional details regarding the use and operation of this and other **LapSnapper** products, please visit the official **LapSnapper** website at www.lapsnapper.com.

All rights reserved.

1. CONTENTS

1. CONTENTS....2

2. VERSION HISTORY....3

3. OVERVIEW....4

4. SOCKET MESSAGE DESCRIPTION....7

4.1. SOCKET MESSAGE FRAME....7
4.2. EXAMPLE: SENDING SOCKET MESSAGES....7
4.3. EXAMPLE OF A BINARY-ENCODED SOCKET MESSAGE....9

5. CONNECTION MANAGEMENT PROTOCOL....12

5.1. HANDSHAKING BETWEEN SERVER AND CLIENT....12
5.2. HEARTBEATS....15
5.3. DISCONNECTING THE CLIENT....15

6. TIME MEASUREMENT PROTOCOL....16

6.1. GET DECODER TIME....16
6.2. TIME EVENTS....18

7. REMOTE CONTROL PROTOCOL....19

7.1. STOP SESSION....19
7.2. EXIT SESSION....19
7.3. START SESSION....20
7.4. STARTMATIC STATUS....21
7.5. SESSION INFORMATION....22
7.6. RACE TRACK SETTINGS....23

8. INFORMATION ELEMENTS....24

8.1. LAPSNAPPER PRODUCT ID DATA FIELD....24
8.2. SYSTEM STATE DATA FIELD....24
8.3. SESSION PARAMETERS DATA FIELD....25

9. FREQUENTLY ASKED QUESTIONS (FAQ)....26

10. TROUBLESHOOTING TIPS....29

11. BEST PRACTICES FOR STABLE OPERATION....32

2. VERSION HISTORY

Version	Date	Description
5.20	December 17, 2025	Race Apps ID assigned
5.20	October 22, 2025	Alpha Timing ID reassigned
5.20	April 4, 2025	Overview chapter is updated
5.20	April 2, 2025	Document layout is unified with other LapSnapper documentation and general update of document
5.20	Jan 23, 2025	Changes: - Chapter 4.1 Connect to the LapSnapper Socket Server: password is corrected
5.20	Oct 29, 2024	Changes: - 2 notes added to the 5.2 Time Events Message - Editorial changes of the 6.3 Start Session Message
5.20	Feb 3, 2021	ID for Volare is added
5.20	Dec 9, 2020	More examples of the binary coded messages are added
5.20	Dec 7, 2020	Editorial changes
5.20	Nov 20, 2020	Message header is object and productID is renamed to lapSnapperProductID
5.20	Nov 16, 2020	Examples are added
5.20	Feb 14, 2020	Session parameters data field is updated
5.20	Feb 10, 2020	Megatiming product ID is added
5.20	Feb 3, 2020	Remote control interface is added
2.00	Oct 19, 2018	JSON based message coding is introduced
1.00	Jan 10, 2017	LapSnapper API is introduced

3. OVERVIEW

This document describes the **LapSnapper Application Programming Interface**, which enables third-party applications to communicate with the **LapSnapper Lap Timing System**.

The LapSnapper system acts as a **socket server**, while external applications function as **socket clients**. Communication is based on a binary frame structure, containing **JSON-encoded messages**, transmitted over a **TCP connection**.

The purpose of this document is to define the communication protocol, message formats, connection handling, time measurement processes, and remote control operations.

By following the specifications described here, third-party developers can integrate their applications with LapSnapper for tasks such as:

- Receiving lap and sector times
- Managing timing sessions
- Controlling session flow (start, stop, exit)
- Synchronizing timing information
- Sending session and track configuration data

Key Features of the Interface

- **Socket Server Communication**: LapSnapper listens for incoming client connections on a specified TCP port.
- **Binary-Framed JSON Messages**: All communication is structured with a consistent header and JSON payload.
- **Real-Time Timing Events**: Clients receive real-time data for transponder time measurements.
- **Remote Session Control**: Clients can remotely start, stop, and manage sessions using authorized control messages.
- **Connection Maintenance**: Heartbeat messages are used to maintain and monitor the active socket connection.

Target Audience

This document is intended for **software developers** and **system integrators** who are building applications that interface with the **LapSnapper Lap Timing System**.

Developers should have a working knowledge of:

- TCP/IP socket programming
- JSON data structures
- Basic understanding of lap time measurement systems

Please see the LapSnapper System Diagram below:

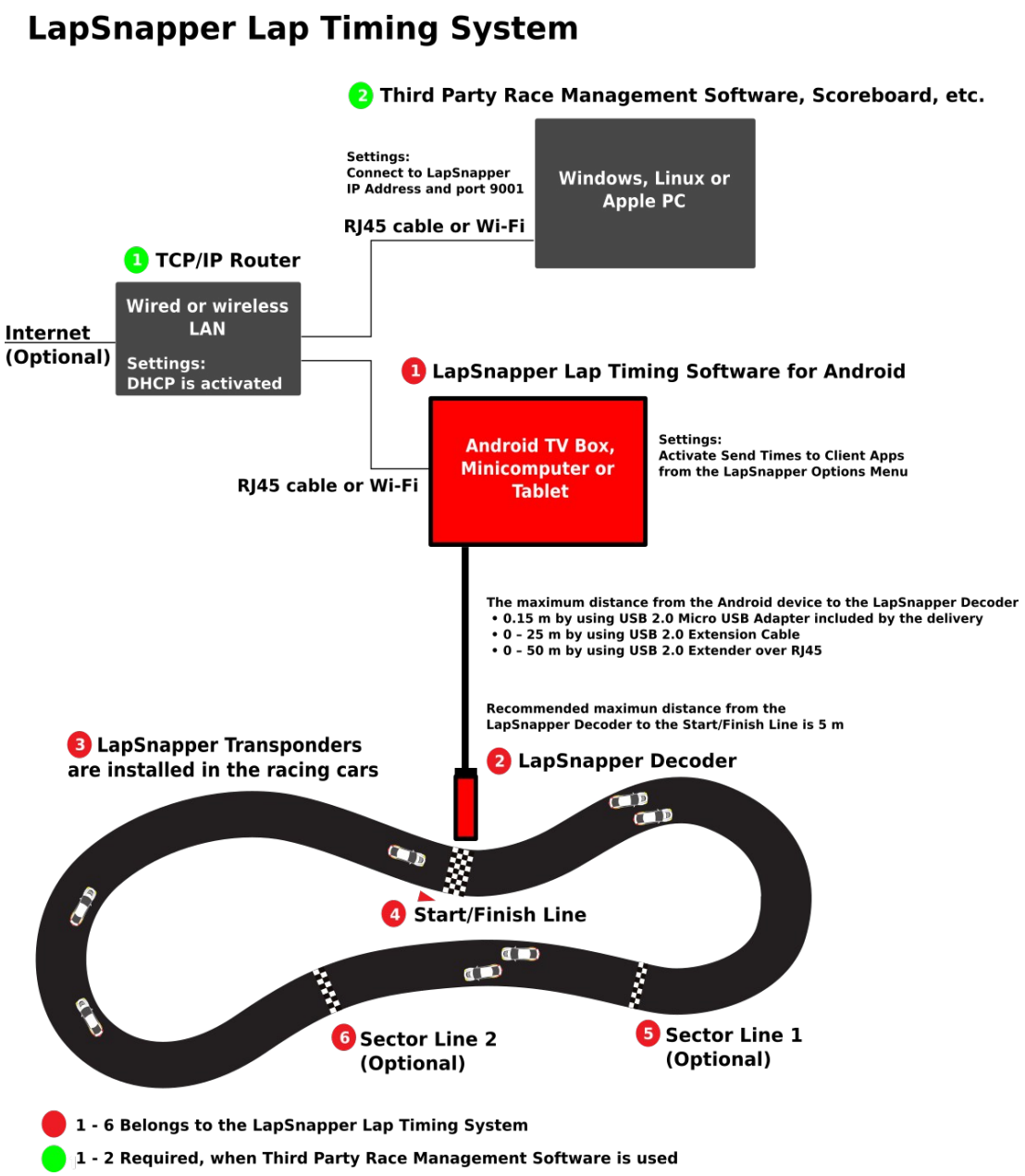

Figure 1: LapSnapper System Diagram

4. SOCKET MESSAGE DESCRIPTION

4.1. SOCKET MESSAGE FRAME

Messages are encoded using **JSON** and packed into a socket message frame with the following structure:

byte 1		byte 6 + n
2 bytes - Message ID	4 bytes - Message length (= n)	n bytes - Message decoded by using JSON

⚠ The **Message ID** and **Message Length** fields are encoded in **little-endian** format.

⚠ The **payload** is a standard JSON object, encoded as a binary byte array.

4.2. EXAMPLE: SENDING SOCKET MESSAGES

Step 1: Create JSON Object

```
// Create JSON object
JSONObject message = new JSONObject();
try {
    ////////////////////////////////////////////////////////
    // Build message header fields
    buildMessageHeader(
        message,
        MSG_LAPSNAPPER_SERVER_API_GET_SERVER_INFO,
        productID,
        supportedServerVersion
    );

    ////////////////////////////////////////////////////////
    // Serialize password array
    JSONArray passwords = new JSONArray();
    for (int i = 0; i < password.length; i++) {
        passwords.put(password[i]);
    }
    message.put("password", passwords);

} catch (JSONException e) {
    CError.exception(e);
}
```

Step 2: Build the Socket Message

```
////////////////////////////////////////////////////////////
// Convert JSON message to byte array (UTF-8 encoding)
final byte[] payload = message.toString().getBytes(StandardCharsets.UTF_8);

// Calculate total message length (header + payload)
int messageLength = 6 + payload.length;
byte[] serializedMessage = new byte[messageLength];

// Serialize socket message header
int index = 0;
index = CDataSerializer.serializeSocketMessageFrameHeader(
    serializedMessage,
    index,
    messageID,
    payload.length
);

// Serialize payload into the message
CDataSerializer.serializeByteArray(serializedMessage, index, payload);
```

Step 3: Send the Socket Message

```
// Send the serialized message over the socket
writeData(serializedMessage);
```

4.3. EXAMPLE OF A BINARY-ENCODED SOCKET MESSAGE

HANDSHAKING BETWEEN CLIENT AND SERVER (Used Client ID 1000000)

11:04:59.680 <Tag> Client <I> GET SERVER INFORMATION

- Message Id 0x1770 = 6000 and length 0x0000006E = 110

70 17 6E 00 00 00 7B 22 6D 65 73 73 61 67 65 48 65 61 64 65 72 22 3A 7B 22 6D 65 73 73 61 67 65 49 44 22 3A 36 30 30 30 2C 22 6C 61 70 53 6E 61 70 70 65 72 50 72 6F 64 75 63 74 49 44 22 3A 31 30 30 30 30 30 30 2C 22 73 65 72 76 65 72 56 65 72 73 69 6F 6E 22 3A 35 32 30 30 30 30 7D 2C 22 70 61 73 73 77 6F 72 64 22 3A 5B 31 2C 32 2C 33 2C 34 5D 7D

11:04:59.705 <Tag> Server <I> SERVER INFORMATION (0x1771 = 6001)

- Message Id 0x1771 = 6001 and length 0x000000D8 = 216

71 17 D8 00 00 00 7B 22 6D 65 73 73 61 67 65 48 65 61 64 65 72 22 3A 7B 22 6D 65 73 73 61 67 65 49 44 22 3A 36 30 30 31 2C 22 6C 61 70 53 6E 61 70 70 65 72 50 72 6F 64 75 63 74 49 44 22 3A 36 30 2C 22 73 65 72 76 65 72 56 65 72 73 69 6F 6E 22 3A 35 32 30 30 30 30 7D 2C 22 64 65 63 6F 64 65 72 49 6E 66 6F 22 3A 7B 22 70 72 6F 64 75 63 74 49 44 22 3A 31 2C 22 76 65 72 73 69 6F 6E 4E 75 6D 62 65 72 22 3A 35 31 30 2C 22 64 65 76 69 63 65 54 79 70 65 22 3A 32 2C 22 64 65 76 69 63 65 49 44 22 3A 31 30 30 32 32 2C 22 64 65 63 6F 64 65 72 4D 6F 64 65 22 3A 31 2C 22 64 65 63 6F 64 65 72 53 74 61 74 75 73 22 3A 31 7D 2C 22 73 79 73 74 65 6D 53 74 61 74 65 22 3A 30 7D

NORMAL OPERATION (When Connection is Active)

11:04:59.917 <Tag> Server <I> HEATBEAT

- Message Id 0x1773 = 6003 and length 0x00000054 = 84

73 17 54 00 00 00 7B 22 6D 65 73 73 61 67 65 48 65 61 64 65 72 22 3A 7B 22 6D 65 73 73 61 67 65 49 44 22 3A 36 30 30 33 2C 22 6C 61 70 53 6E 61 70 70 65 72 50 72 6F 64 75 63 74 49 44 22 3A 36 30 2C 22 73 65 72 76 65 72 56 65 72 73 69 6F 6E 22 3A 35 32 30 30 30 30 7D 7D

11:05:01.918 <Tag> Server <I> HEATBEAT

- Message Id 0x1773 = 6003 and length 0x00000054 = 84

73 17 54 00 00 00 7B 22 6D 65 73 73 61 67 65 48 65 61 64 65 72 22 3A 7B 22 6D 65 73 73 61 67 65 49 44 22 3A 36 30 30 33 2C 22 6C 61 70 53 6E 61 70 70 65 72 50 72 6F 64 75 63 74 49 44 22 3A 36 30 2C 22 73 65 72 76 65 72 56 65 72 73 69 6F 6E 22 3A 35 32 30 30 30 30 7D 7D

11:05:03.921 <Tag> Server <I> HEATBEAT

- Message Id 0x1773 = 6003 and length 0x00000054 = 84

73 17 54 00 00 00 7B 22 6D 65 73 73 61 67 65 48 65 61 64 65 72 22 3A 7B 22 6D 65 73 73 61 67 65 49 44 22 3A 36 30 30 33 2C 22 6C 61 70 53 6E 61 70 70 65 72 50 72 6F 64 75 63 74 49 44 22 3A 36 30 2C 22 73 65 72 76 65 72 56 65 72 73 69 6F 6E 22 3A 35 32 30 30 30 30 7D 7D

11:05:23.166 <Tag> Client <I> START SESSION (Remote control password: “1234”)

- Message Id 0x1902 = 6402 and length 0x00000233 = 563

02 19 33 02 00 00 7B 22 6D 65 73 73 61 67 65 48 65 61 64 65 72 22 3A 7B 22 6D 65 73 73 61 67 65 49 44 22 3A 36 34 30 32 2C 22 6C 61 70 53 6E 61 70 70 65 72 50 72 6F 64 75 63 74 49 44 22 3A 31 30 30 30 30 30 30 2C 22 73 65 72 76 65 72 56 65 72 73 69 6F 6E 22 3A 35 32 30 30 30 30 7D 2C 22 70 69 6E 43 6F 64 65 22 3A 22 31 32 33 34 22 2C 22 73 65 73 73 69 6F 6E 50 61 72 61 6D 65 74 65 72 73 22 3A 7B 22 73 65 73 73 69 6F 6E 54 79 70 65 22 3A 30 2C 22 74 72 69 67 67 65 72 4C 69 6E 65 43 6F 75 6E 74 22 3A 31 2C 22 73 74 61 72 74 4C 69 6E 65 22 3A 30 2C 22 73 74 6F 70 54 79 70 65 22 3A 31 2C 22 73 74 6F 70 56 61 6C 75 65 22 3A 2D 31 2C 22 63 6F 6E 74 69 6E 75 6F 75 73 53 65 73 73 69 6F 6E 22 3A 66 61 6C 73 65 2C 22 70 65 72 73 6F 6E 61 6C 53 74 6F 70 70 69 6E 67 43 72 69 74 65 72 69 61 22 3A 66 61 6C 73 65 2C 22 72 65 73 75 6C 74 73 41 72 65 4F 72 64 65 72 65 64 41 67 61 69 6E 73 74 54 6F 74 61 6C 54 69 6D 65 22 3A 66 61 6C 73 65 2C 22 73 65 73 73 69 6F 6E 53 74 6F 70 53 74 61 72 74 73 57 68 65 6E 4C 65 61 64 65 72 52 65 61 63 68 65 73 53 74 6F 70 56 61 6C 75 65 22 3A 66 61 6C 73 65 7D 2C 22 74 72 61 63 6B 53 65 74 74 69 6E 67 73 22 3A 7B 22 6D 69 6E 69 6D 75 6D 4C 61 70 54 69 6D 65 45 6E 61 62 6C 65 64 22 3A 66 61 6C 73 65 2C 22 6D 69 6E 69 6D 75 6D 4C 61 70 54 69 6D 65 22 3A 31 35 30 30 30 2C 22 6D 69 6E 69 6D 75 6D 53 65 63 74 6F 72 31 54 69 6D 65 22 3A 35 30 30 30 2C 22 6D 69 6E 69 6D 75 6D 53 65 63 74 6F 72 32 54 69 6D 65 22 3A 35 30 30 30 2C 22 6D 69 6E 69 6D 75 6D 53 65 63 74 6F 72 33 54 69 6D 65 22 3A 35 30 30 30 2C 22 6D 61 78 69 6D 75 6D 4C 61 70 54 69 6D 65 45 6E 61 62 6C 65 64 22 3A 66 61 6C 73 65 2C 22 6D 61 78 69 6D 75 6D 4C 61 70 54 69 6D 65 22 3A 32 34 30 30 30 30 7D 7D

11:05:25.707 <Tag> Server <I> STARTMATIC STATUS (Remote control password: “1234”)

- Message Id 0x1903 = 6403 and length 0x000000A7 = 167

03 19 A7 00 00 00 7B 22 6D 65 73 73 61 67 65 48 65 61 64 65 72 22 3A 7B 22 6D 65 73 73 61 67 65 49 44 22 3A 36 34 30 33 2C 22 6C 61 70 53 6E 61 70 70 65 72 50 72 6F 64 75 63 74 49 44 22 3A 31 30 30 30 30 30 30 2C 22 73 65 72 76 65 72 56 65 72 73 69 6F 6E 22 3A 35 32 30 30 30 30 7D 2C 22 70 69 6E 43 6F 64 65 22 3A 22 31 32 33 34 22 2C 22 73 74 61 72 74 4D 61 74 69 63 53 74 61 74 75 73 22 3A 31 2C 22 73 74 61 72 74 4D 61 74 69 63 44 61 74 61 22 3A 36 30 2C 22 73 74 61 72 74 4D 61 74 69 63 44 61 74 61 32 22 3A 31 7D

11:05:27.341 <Tag> Server <I> STARTMATIC STATUS (Remote control password: “1234”)

- Message Id 0x1903 = 6403 and length 0x000000A8 = 168

03 19 A8 00 00 00 7B 22 6D 65 73 73 61 67 65 48 65 61 64 65 72 22 3A 7B 22 6D 65 73 73 61 67 65 49 44 22 3A 36 34 30 33 2C 22 6C 61 70 53 6E 61 70 70 65 72 50 72 6F 64 75 63 74 49 44 22 3A 31 30 30 30 30 30 30 2C 22 73 65 72 76 65 72 56 65 72 73 69 6F 6E 22 3A 35 32 30 30 30 30 7D 2C 22 70 69 6E 43 6F 64 65 22 3A 22 31 32 33 34 22 2C 22 73 74 61 72 74 4D 61 74 69 63 53 74 61 74 75 73 22 3A 34 2C 22 73 74 61 72 74 4D 61 74 69 63 44 61 74 61 22 3A 31 35 2C 22 73 74 61 72 74 4D 61 74 69 63 44 61 74 61 32 22 3A 2D 31 7D

11:05:30.266 <Tag> Server <I> STARTMATIC STATUS (Remote control password: “1234”)

- Message Id 0x1903 = 6403 and length 0x000000A7 = 167

03 19 A7 00 00 00 7B 22 6D 65 73 73 61 67 65 48 65 61 64 65 72 22 3A 7B 22 6D 65 73 73 61 67 65 49 44 22 3A 36 34 30 33 2C 22 6C 61 70 53 6E 61 70 70 65 72 50 72 6F 64 75 63 74 49 44 22 3A 31 30 30 30 30 30 30 2C 22 73 65 72 76 65 72 56 65 72 73 69 6F 6E 22 3A 35 32 30 30 30 30 7D 2C 22 70 69 6E 43 6F 64 65 22 3A 22 31 32 33 34 22 2C 22 73 74 61 72 74 4D 61 74 69 63 53 74 61 74 75 73 22 3A 36 2C 22 73 74 61 72 74 4D 61 74 69 63 44 61 74 61 22 3A 31 2C 22 73 74 61 72 74 4D 61 74 69 63 44 61 74 61 32 22 3A 2D 31 7D

11:06:46.413 <Tag> Server <I> TIME EVENT

- Message Id 0x17DE = 6110 and length 0x000000E7 = 231

DE 17 E7 00 00 00 7B 22 6D 65 73 73 61 67 65 48 65 61 64 65 72 22 3A 7B 22 6D 65 73 73 61 67 65 49 44 22 3A 36 31 31 30 2C 22 6C 61 70 53 6E 61 70 70 65 72 50 72 6F 64 75 63 74 49 44 22 3A 36 30 2C 22 73 65 72 76 65 72 56 65 72 73 69 6F 6E 22 3A 35 32 30 30 30 30 7D 2C 22 74 72 61 6E 73 70 6F 6E 64 65 72 49 44 22 3A 31 30 30 30 33 2C 22 74 72 61 6E 73 70 6F 6E 64 65 72 49 6E 73 74 61 6E 63 65 22 3A 31 2C 22 74 72 61 6E 73 70 6F 6E 64 65 72 4E 61 6D 65 22 3A 22 4D 61 72 6B 22 2C 22 61 62 73 6F 6C 75 74 65 54 69 6D 65 22 3A 7B 22 74 69 6D 65 53 74 61 6D 70 49 44 22 3A 31 35 37 2C 22 61 62 73 6F 6C 75 74 65 54 69 6D 65 22 3A 31 36 30 37 35 30 34 38 30 33 33 36 31 7D 2C 22 6C 61 70 4C 69 6E 65 22 3A 31 7D

11:06:57.625 <Tag> Server <I> TIME EVENT

- Message Id 0x17DE = 6110 and length 0x00000233 = 234

DE 17 EA 00 00 00 7B 22 6D 65 73 73 61 67 65 48 65 61 64 65 72 22 3A 7B 22 6D 65 73 73 61 67 65 49 44 22 3A 36 31 31 30 2C 22 6C 61 70 53 6E 61 70 70 65 72 50 72 6F 64 75 63 74 49 44 22 3A 36 30 2C 22 73 65 72 76 65 72 56 65 72 73 69 6F 6E 22 3A 35 32 30 30 30 30 7D 2C 22 74 72 61 6E 73 70 6F 6E 64 65 72 49 44 22 3A 31 30 30 30 34 2C 22 74 72 61 6E 73 70 6F 6E 64 65 72 49 6E 73 74 61 6E 63 65 22 3A 31 2C 22 74 72 61 6E 73 70 6F 6E 64 65 72 4E 61 6D 65 22 3A 22 4D 69 63 68 61 65 6C 22 2C 22 61 62 73 6F 6C 75 74 65 54 69 6D 65 22 3A 7B 22 74 69 6D 65 53 74 61 6D 70 49 44 22 3A 31 36 30 2C 22 61 62 73 6F 6C 75 74 65 54 69 6D 65 22 3A 31 36 30 37 35 30 34 38 31 34 35 39 30 7D 2C 22 6C 61 70 4C 69 6E 65 22 3A 31 7D

11:07:06.773 <Tag> Client <I> STOP SESSION (Remote control password: “1234”)

- Message Id 0x1900 = 6400 and length 0x0000006A = 106

00 19 6A 00 00 00 7B 22 6D 65 73 73 61 67 65 48 65 61 64 65 72 22 3A 7B 22 6D 65 73 73 61 67 65 49 44 22 3A 36 34 30 30 2C 22 6C 61 70 53 6E 61 70 70 65 72 50 72 6F 64 75 63 74 49 44 22 3A 31 30 30 30 30 30 30 2C 22 73 65 72 76 65 72 56 65 72 73 69 6F 6E 22 3A 35 32 30 30 30 30 7D 2C 22 70 69 6E 43 6F 64 65 22 3A 22 31 32 33 34 22 7D

5. CONNECTION MANAGEMENT PROTOCOL

5.1. HANDSHAKING BETWEEN SERVER AND CLIENT

The **handshaking process** establishes a valid communication session between the **LapSnapper Socket Server** and a **third-party client application**. This process ensures both compatibility and authorization before further interaction can occur.

To establish communication, client creates a socket connection to the LapSnapper Socket Server using the server's IP address on **port 9001**. Then, send a **"Get LapSnapper Server Information"** request.

Step 1: Send "Get LapSnapper Server Information" Message

JSON	Description

{
"messageHeader": {
"messageID": 6000,
"lapSnapperProductID": Integer,
"serverVersion": 520000
},
"password": [1, 2, 3, 4]
}
	
	Request for server info
	See coding 8.1
	Socket socket server version supported by client v5.20.000
	Array used for authentication

Table 2: Encoding of "Get Server Information" message

Step 2a: Server Response – Success

If the request is successful, the server responds with the **"LapSnapper Server Information"** message:

JSON	Notes

{
"messageHeader": {
"messageID": 6001,
"lapSnapperProductID": Integer,
"serverVersion": 520000
},
"decoderInfo": {
"productID": Integer,
"versionNumber": Integer,
"deviceType": Integer,
"deviceID": Integer,
"decoderMode": Integer,
"decoderStatus": Integer,
},
"systemState": Integer
}
	
	Successful server info response
	See coding 8.1
	Socket Server version number v5.20.000
	
	Example 520000 = 5.20.000
	Decoder type: 1 or 2
	Decoder device ID
	Identifies the decoder role (1 = host, 2 = slave).
	Decoder status
	
	Current system state (See coding 4.3)

Table: Encoding of "LapSnapper Server Information" Message

Once the socket connection between the client and the LapSnapper Socket Server is successfully established and the handshake process is complete, the communication continues as follows:

- The **server sends periodic heartbeat messages** (every 2 seconds) to confirm the connection is alive.
- The **client listens for real-time data**, such as time events and system updates.
- The **client may send control or query messages** (e.g., session commands, time requests) based on application needs.
- Both server and client are expected to handle data asynchronously and maintain proper message parsing according to the binary frame structure.

Step 2b: Server Response – Failure

If the request is invalid (e.g., unsupported version or server type), the server responds with the **"LapSnapper Server Information Failed"** message:

JSON	Notes

{
"messageHeader": {
"messageID": 6002,
"lapSnapperProductID": Integer,
"serverVersion": 520000
},
"status": 1
}
	
	Indicates failure
	See coding 8.1
	Socket Server version (5.20.000)
	
	Error code for unsupported version or invalid credentials

Table 3: Enoding of "LapSnapper Server Information Failed" Message

Step 2c: Server-Initiated Disconnection

If the password is incorrect or an invalid message is received, the LapSnapper Socket Server will **disconnect the client** immediately.

5.2. HEARTBEATS

To maintain an active connection, the **LapSnapper Socket Server** sends **"Heartbeat"** messages to all connected clients every **2 seconds**. These messages confirm that the connection is alive and synchronized.

JSON	Notes

{
"messageHeader": {
"messageID": 6003,
"lapSnapperProductID": Integer,
"serverVersion": 520000
}
}
	Identifies the message as a heartbeat
	See coding 8.1
	Socket socket server version supported by client (5.20.000)

Table 4: Encoding of "Heartbeat" Message

5.3. DISCONNECTING THE CLIENT

To terminate the connection, the client should gracefully **disconnect the socket** when communication with the LapSnapper Socket Server is no longer required.

This involves:

- Closing the socket connection properly.
- Releasing any associated resources.
- Optionally notifying the server before disconnecting (if supported by the protocol).

Proper disconnection helps maintain server stability and ensures clean session management.

6. TIME MEASUREMENT PROTOCOL

6.1. GET DECODER TIME

The client can request the current decoder time from the LapSnapper Socket Server. This is useful for synchronization and time-based event handling.

Step 1: Send "Get Decoder Time" Message

JSON	Notes

{
"messageHeader": {
"messageID": 6100,
"lapSnapperProductID": Integer,
"serverVersion": 520000
}
}
	Request to retrieve the current decoder time
	See coding 8.1
	Socket socket server version supported by client (5.20.000)

Table 5: Encoding of "Get Decoder Time" Message

Step 2a: Server Response – Success

If the request is successful, the server responds with a **"Decoder Time"** message:

JSON	Notes

{
"messageHeader": {
"messageID": 6101,
"lapSnapperProductID": Integer,
"serverVersion": 520000
},
"decoderTime": Long
}
	Indicates successful retrieval of decoder time
	See coding 8.1
	Socket Server version (5.20.000)
	
	The current decoder time, in **milliseconds**

Table 6: Encoding of "Decoder Time" Message

Step 2b: Server Response – Failure

If the decoder time is not available, the server responds with a **"Decoder Time Failed"** message:

JSON	Notes

{
"messageHeader": {
"messageID": 6102,
"lapSnapperProductID": Integer,
"serverVersion": 520000
}
}
	Indicates failure to retrieve decoder time
	See coding 8.1
	Socket Server version (5.20.000)

Table 7: Encoding of "Decoder Time Failed" Message

6.2. TIME EVENTS

The **LapSnapper Socket Server** sends a **"Time Event"** message to connected clients whenever a new timing event is registered for a transponder.

JSON	Notes

{
"messageHeader": {
"messageID": 6110,
"lapSnapperProductID": Integer,
"serverVersion": 520000
},
"transponderID": Integer,
"transponderInstance": Integer,
"transponderName": String,
"absoluteTime": {
"timeStampID": Integer,
"absoluteTime": Long
},
"lapLine": Integer
}
	Identifies the message as a Time Event
	See coding 8.1
	Socket Server version (5.20.000)
	
	The unique identifier of the transponder
	
	Optional label or name for the transponder
	
	A unique identifier for the timestamp
	Timestamp value in **milliseconds**
	
	This message is only available when a **timing session is active** and **sector timing** is enabled:
	1 if lap line and 0 if sector line:
	• `1` indicates a **lap line**
	• `0` indicates a **sector line**

Table 8: Encoding of "Time Event" Message

Additional Notes:

- **Multiple Time Events:**

  It is possible to receive **multiple time event messages in quick succession**. This can happen if:

  - Sector timing is enabled (multiple timing lines per lap).
  - The transponder had previously missed sending data, and buffered messages are being sent.

- **Message Order Not Guaranteed:**

  The order of time events is **not strictly chronological**. In rare cases—such as temporary disconnection of the decoder's USB interface—**older timestamps may arrive after newer ones**. Client applications must be able to handle out-of-order time events.

7. REMOTE CONTROL PROTOCOL

The **Remote Control Interface** allows external applications to control the session state of the LapSnapper system. This interface can be enabled or configured from the LapSnapper application via **Options** → **Remote Control Settings**.

7.1. STOP SESSION

To stop an active session, send a **"Stop Session"** message to the LapSnapper Socket Server.

JSON	Notes

{
"messageHeader": {
"messageID": 6400,
"lapSnapperProductID": Integer,
"serverVersion": 520000
},
"pinCode": String
}
	Identifies the message as a request to stop the session
	See coding 8.1
	Socket Server version (5.20.000)
	
	A valid PIN code required to authorize the operation

Table 9: Encoding of "Stop Session" Message

7.2. EXIT SESSION

To completely exit the session and reset the system state, send an **"Exit Session"** message to the LapSnapper Socket Server.

JSON	Notes

{
"messageHeader": {
"messageID": 6401,
"lapSnapperProductID": Integer,
"serverVersion": 520000
},
"pinCode": String
}
	Identifies the message as a request to exit the session
	See coding 8.1
	Socket Server version (5.20.000)
	
	A valid PIN code required to authorize the operation

Table 10: Encoding of "Exit Session" Message

7.3. START SESSION

To initiate a new timing session, send a **"Start Session"** message to the LapSnapper Socket Server. This message includes session parameters and track-specific settings to configure the behavior of the session.

JSON	Notes
{	
"messageHeader": {	Identifies the message as a "Start Session" request
"messageID": 6402,	See coding 8.1
"lapSnapperProductID": Integer,	Socket Server version (5.20.000)
"serverVersion": 520000	A valid PIN code required to authorize the operation
},	See coding 8.3
"pinCode": String,	Session type
"sessionParameters": {	Sector count
"sessionType": Integer,	Start line
"triggerLineCount": Integer,	Stop type
"startLine": Integer,	Stop value
"stopType": Integer,	Continuous session
"stopValue": Integer,	Personal stopping criteria
"continuousSession": Boolean,	Results of the qualifying session are ordered against total time
"personalStoppingCriteria": Boolean,	Session is stopped when the leader has finished
"resultsAreOrderedAgainstTotalTime": Boolean,	
"sessionStopStartsWhenLeader ReachesStopValue": Boolean	
},	
"trackSettings": {	
"minimumLapTimeEnabled": Boolean,	Minimum lap time is enabled
"minimumLapTime": Integer,	Minimum lap time
"minimumSector1Time": Integer,	Minimum sector 1 time
"minimumSector2Time": Integer,	Minimum sector 2 time
"minimumSector3Time": Integer,	Minimum sector 3 time
"maximumLapTimeEnabled": Boolean,	Maximum lap time is enabled
"maximumLapTime": Integer	Maximum lap time
}	
}	

Table 11: Encoding of "Start Session" Message

7.4. STARTMATIC STATUS

The **"StartMatic Status"** message is used to update or control the StartMatic functionality on the LapSnapper Socket Server. This message includes status flags and associated timing or control data.

To send a status update, the client must transmit a **"StartMatic Status"** message to the server.

JSON	Notes

{
"messageHeader": {
"messageID": 6403,
"lapSnapperProductID": Integer,
"serverVersion": 520000
},
"pinCode": String,
"startMaticStatus": Integer,
"startMaticData": Long,
"startMaticData2": Long
}
	Identifies the message as a StartMatic status update
	See coding 8.1
	Socket Server version (5.20.000)
	
	A valid PIN code required to authorize the operation
	See StartMatic Statuses from table 13

Table 12: Encoding of "StartMatic Status" Message

StartMatic Statuses:

Value	Notes
1 startMaticData startMaticData2	StartMatic is opened - StartMatic Delay - StartMatic Mode
2	StartMatic is closed
3	StartMatic is canceled
4 startMaticdata	StartMatic Delay - StartMatic Delay
5 startMaticData	StarMatic Mode - StartMatic Mode
6	StartMatic countdown is started

Table 13: Encoding of StartMatic Statuses

7.5. SESSION INFORMATION

The **"Session Information"** message is used to send detailed information about the current session—including session name, driver list, and transponder configurations—to the LapSnapper Socket Server.

This message is typically used before or at the start of a session to ensure the server has accurate participant data.

JSON	Notes

{
"messageHeader": {
"messageID": 6404,
"lapSnapperProductID": Integer,
"serverVersion": 520000
},
"pinCode": String,
"sessionInformation": {
"sessionName": String,
"allTranspondersActive": Boolean,
"driverInformation": {
[{
"driverNumber": Integer,
"driverName": String,
"transponderID": {
"transponderID": Integer,
"transponderInstance": Integer
},
"transponderEnabled": Boolean
}
]
}
}
	Identifies the message as a Session Information update
	See coding 8.1
	Socket Server version (5.20.000)
	
	A valid PIN code required to authorize the operation
	Session name
	All transponders are active
	
	Driver number
	Driver name
	
	Transponder ID
	Transponder Instance (Default = 1)
	
	Transponder is enabled

Table 14: Encoding of "Session Information" Message

7.6. RACE TRACK SETTINGS

The **"Race Track Settings"** message is used to configure track-related time limits, such as minimum and maximum lap or sector times. These settings help filter out invalid lap data and ensure accurate timekeeping.

The client must send this message to the **LapSnapper Socket Server** to apply the desired timing constraints for the session.

JSON	Notes

{
"messageHeader": {
"messageID": 6405,
"lapSnapperProductID": Integer,
"serverVersion": 520000
},
"pinCode": String,
"trackSettings": {
"minimumLapTimeEnabled": Boolean,
"minimumlapTime": Integer,
"minimumSector1Time": Integer,
"minimumSector2Time": Integer,
"minimumSector3Time": Integer,
"maximumLapTimeEnabled": Boolean,
"maximumLapTime": Integer
}
}
	
	Identifies the message as a request to set race track timing parameters
	See coding 8.1
	Socket Server version (5.20.000)
	
	A valid PIN code required to authorize the operation
	Minimum lap time is enabled
	Minimum lap time
	Minimum sector 1 time
	Minimum sector 2 time
	Minimum sector 3 time
	Maximum lap time is enabled
	Maximum lap time

Table 15: Encoding of "Race Track Settings" message

8. INFORMATION ELEMENTS

8.1. LAPSNAPPER PRODUCT ID DATA FIELD

Value	Description
50 200 10000 10001 10002 10003 10004 10005 10006 > 1000000	LapSnapper Product IDs: - LapSnapper Lap Timing Software for Android - LapSnapper Client&Monitoring Software for Android - Kart timer (www.karttimer.fi) - RaceFacer (www.racefacer.com) - RC Timing (www.rc-timing.ch) - Megatiming (www.megatiming.se) - Volare (www.volarehq.com) - Alpha Timing (www.alphatiming.co.uk) - Race Apps (race-apps.com) - Free for all applications

Table 16: Encoding of product ID data field

8.2. SYSTEM STATE DATA FIELD

Value	Description
0 1 2 3 4 5 6 7 8 9 10 -1	System state - Main menu state - Session options dialog state - Timing state - Timing recovery dialog state - Results dialog state - Results browsing state - Results browsing save results dialog state - Settings dialog state - Save results restart dialog state - Save results dialog state - Automatic session startup state - Unknown state

Table 17: Encoding of System State data field

8.3. SESSION PARAMETERS DATA FIELD

Value	Description
0 1 2	Session type - Practice - Qualification - Race
1, 2 or 3	Sector count - 1, 2 or 3 sectors
0, 1 or 2	Start line - Start/Finish line is 1st, 2nd or 3rd sector line
1 2 3	Stop type - Manual - Session time - Lap count
N/A X seconds X laps	Stop value - Manual stop: N/A - Session time stop: Session time in seconds - Lap count stop: lap count
False True	Continuous session (Practice and qualification sessions) - Disabled - Enabled
False True	Personal stopping criteria (Practice and qualification sessions) - Disabled - Enabled
False True	Time order of the qualification session - Results are ordered against the best lap time - Results are ordered against the total time
False True	Session is stopped when the leader has finished - Session is stopped when the session time has ended or lap count is reached - Session is stopped when the leader has finished

Table 18: Encoding of Session Parameters data field

9. FREQUENTLY ASKED QUESTIONS (FAQ)

1. What is the LapSnapper Socket Server?

The LapSnapper Socket Server is part of the LapSnapper Lap Timing Software. It enables third-party applications to connect via TCP sockets and exchange timing and control information using a structured binary protocol with JSON-encoded messages.

2. What kind of data format is used for communication?

Each message uses a **binary frame**:

- 2 bytes for Message ID (little-endian)
- 4 bytes for Message Length (little-endian)
- n bytes for a **JSON-encoded payload**

All timing, control, and status information is sent inside the JSON payload.

3. How do I establish a connection with the LapSnapper Socket Server?

1. Open a **TCP connection** to the server's IP address on **port 9001**.
2. Send a **"Get Server Information"** request to perform a handshake.
3. Wait for a successful response before sending further messages.

4. What happens if the server version does not match?

If the client's supported server version does not match the LapSnapper server version, the server responds with a **"Server Information Failed"** message and may terminate the connection.

5. What are heartbeat messages and why are they important?

The server sends a **heartbeat message** every **2 seconds** to confirm the connection is still alive.

Clients should monitor these heartbeats to detect disconnections or network issues.

6. How is sector timing handled?

Sector timing is managed by placing multiple **timing lines** (Max. 3) around the track.

Each time a transponder crosses a finish line, the server sends a **Time Event** messages with the timestamp information.

7. Can multiple Time Event messages arrive at the same time?

Yes. If sector timing is active, or if there were temporary transmission delays, the server may send **multiple Time Event messages at once**. Clients must be prepared to process multiple messages quickly and handle possible out-of-order timestamps.

8. How do I remotely start, stop, or exit a session?

By sending specific **Remote Control messages** with a valid **PIN code**:

- **Start Session** (Message ID 6402)
- **Stop Session** (Message ID 6400)
- **Exit Session** (Message ID 6401)

Each action requires proper authorization via the PIN code.

9. What happens if the PIN code is invalid?

If an incorrect PIN code is provided, the server will reject the request, and in some cases, it may **disconnect the client** for security reasons.

10. Is there a limit to the number of connected clients?

The maximum number of concurrent clients is determined by the server's configuration and available system resources.

LapSnapper typically supports **multiple clients simultaneously**, but each must perform the proper handshake procedure.

11. How should I disconnect properly from the server?

Clients should **gracefully close** the socket connection when communication is no longer needed to free server resources and ensure a clean shutdown.

12. What should I do if I lose the connection?

If the connection is lost (e.g., missing heartbeats or network issues), the client should:

- Attempt to **reconnect** after a short delay.
- **Re-initiate the handshake** procedure.
- **Resynchronize** session state if necessary.

10. TROUBLESHOOTING TIPS

This section provides solutions to common issues that may occur when developing or integrating with the LapSnapper Socket Server.

1. Connection Fails

Symptoms:

- Unable to connect to server.
- TCP socket connection is refused.

Possible Causes:

- Incorrect server IP address or port (default is **9001**).
- Server is not running or listening.
- Firewall blocking the connection.

Solutions:

- Verify the server's IP address and port number.
- Ensure the LapSnapper software is running and configured to accept socket connections.
- Check firewall or network settings to allow traffic on the correct port.

2. No Response After Connection

Symptoms:

- TCP connection succeeds but no messages are received.

Possible Causes:

- Handshake message ("Get Server Information") was not sent.
- Message format is incorrect.

Solutions:

- Always send the initial handshake message after connecting.
- Ensure the message frame (Message ID, Length, JSON payload) follows the required binary structure and little-endian encoding.

3. Heartbeats Not Received

Symptoms:

- No heartbeat messages from the server.
- Unexpected disconnects.

Possible Causes:

- Connection not fully established (handshake incomplete).
- Network interruptions.

Solutions:

- Confirm that the handshake completed successfully.
- Check for network reliability issues (e.g., unstable Wi-Fi, VPNs).
- Implement a timeout mechanism in the client to detect missed heartbeats and reconnect automatically.

4. Time Event Messages Are Missing or Out of Order

Symptoms:

- Missing sector or lap times.
- Older timestamps received after newer ones.

Possible Causes:

- Temporary decoder communication issues (e.g., USB disconnection).
- Buffering or transmission delays in the network.

Solutions:

- Always **sort received timestamps by timestamp ID** if order is critical.
- Implement logic to detect and handle **out-of-order events** gracefully.

5. Server Disconnects Client Unexpectedly

Symptoms:

- Server closes connection after message is sent.

Possible Causes:

- Invalid or unsupported message format.
- Incorrect PIN code provided.
- Server-side timeout or protocol violation.

Solutions:

- Validate that all outgoing messages match the correct structure.
- Verify that the PIN code is correct and valid.
- Handle socket disconnections by retrying with correct parameters after a short delay.

6. Server Version Mismatch

Symptoms:

- Server responds with a "Server Information Failed" message.

Possible Causes:

- Client is using an unsupported server version.

Solutions:

- Update the client application to support the server version (e.g., 5.20.000 or later).
- Confirm the `serverVersion` field is correctly populated in handshake and control messages.

11. BEST PRACTICES FOR STABLE OPERATION

- Always **validate message formats** before sending.
- **Monitor heartbeat messages** to detect connection issues early.
- **Reconnect automatically** if the connection is lost.
- **Handle multiple Time Event messages** arriving simultaneously.
- **Log communication errors** and important events for easier debugging.