Open LED Race Network Protocol

Table of Contents

Terminology

In the following part of this doc the terms “**OLR**”, “**Device**”, “**RaceDevice**”, “**Racetrack**” will indicate the same thing:

> **A network-connected Open Led Race Device + an OLR Controller** ([Computer running Network Client Software] + [Arduino running Software + Led Strip]).

“**Race**” indicates a set of **OLR**s virtually connected to create a ***Relay Race***

“**OLR Network**” indicates the Network infrastructure where the Devices connect to partecipate to a Relay Race.

The present document describes the **Application Protocol** used in the OLR Network. The OLRNetwork protocol uses MQTT as “transport” for its messages (MQTT payloads).

“***OLR Network Software***”, “***OLR Client Software***” or simply “**Network *Client***” indicates the software running on the Computer that “enables” a RaceDevice to communicates with the Network.

MQTT Basics

Retained Messages and Last Will

Retained Message

A retained message is a normal MQTT message with the retained flag set to true. The broker stores the last retained message and the corresponding QoS for that topic. Each client that subscribes to a topic pattern that matches the topic of the retained message ***receives the retained message immediately after they subscribe***. The broker stores only one retained message per topic.

In other words, a retained message on a topic is the ***last known good value***. The retained message doesn’t have to be the last value, but it must be the last message with the retained flag set to true.

This feature is used in the “Open Led Race Network”

Last Will and Testament (LWT)

The Last Will and Testament feature provides a way for clients to respond to ungraceful disconnects in an appropriate way.

In MQTT, you use the Last Will and Testament (LWT) feature to notify other clients about an ungracefully disconnected client. **Each client can specify its last will message when it connects to a broker. The last will message is a normal MQTT message with a topic, retained message flag, QoS, and payload**. The broker stores the message until it detects that the client has disconnected ungracefully. In response to the ungraceful disconnect, the broker sends the last-will message to all subscrobed clients of the last-will message topic. If the client disconnects gracefully with a correct DISCONNECT message, the broker discards the stored LWT message.

Retained Message + Last Will and Testament

In real-world scenarios, LWT is often combined with **retained messages** to store the state of a client on a specific topic.

> For example, **Client1** first sends a CONNECT message to the broker with a lastWillMessage that has "*Offline*" as the payload, the lastWillRetain flag set to true, and the lastWillTopic set to *client1/status*. Next, the client PUBLISH a message with the payload "*Online*" and the retained flag set to true to the same topic (*client1/status*). As long as client1 stays connected, newly-subscribed clients to the client1/status topic receive the "*Online*" retained message. If client1 disconnects unexpectedly, the broker publishes the LWT message with the payload "*Offline*" as the new retained message. Clients that subscribe to the topic while Client1 is offline, receive the LWT retained message (”*Offline*”) from the broker.

This pattern of retained messages keeps other clients up to date on the current status of Client1 on a specific topic.

Note:

> *In the following part of this Doc the terms "***Pub***" and "***Sub***" stands for Publish and Subscribe to Topics.*

Source:

- https://www.hivemq.com/blog/mqtt-essentials-part-8-retained-messages/
- https://www.hivemq.com/blog/mqtt-essentials-part-9-last-will-and-testament/

MQTT Infrastructure

OLR Network fundamentals

The whole system is based on a simple assumption:

> At any moment in time, [Retained Messages] ‘*stored*’ in topics described below, contains a complete description of the OLR Network Status:
>
> - Connected clients (Clients list)
> - Active races (Race List)
>   - Participants

When a user “turn on” its Client and **connects** to the OLR Network, receives Retained Messages describing the current situation of any other ***NetworkClient*** (OLR device) and any “***Currently Active Race***”.

The Client does not rely on anything else to reconstruct the network situation at startup.

The real situation is more complicated than this. For example, think about a Client participating to a Race in “Racing” status (cars are moving here or in another participating circuit). **If it disconnects unexpectedly** (network problem) you will have a Race where one of the participating Racetracks disappear...

The current test implementaton of the OLRNetwork Client does not manage these situations.

MQTT Topics Set used in the OLRNetwork

Topics used in the current implementation can be diveded in two sets:

- Devices: Topics related to Devices connected to the Network (Status, etc)
- Races: Topics related to Races currently defined in the Network

Topics related to Devices

Devices currently connected to the OLRNetwork

OLRN Devices Topic Root = “**OLR/basePool/device**”

Implemented: *(**N**)ot yet (**P**)artial (**Y**)es*

*OLR/basePool/device/***status/<*DeviceId*>** **(*) Device List** • Each OLR **Publish** its Status on the specific “status/<Device*Id*>” sub-topic • Each OLR **Sub** to device*Root/***status**/+ *to receive updates for **Devices list** (who is online)* • ***Is the topic used by the client to specify the Last Will message=Offline when it connects to a broker.***	**Y**
*OLR/basePool/device/***Recv/<*DeviceId*>** • Each OLR **Sub** to **its own <Device*Id*>** topic • Other OLR uses the Recv/<Device*Id*> topic to send messages only to one OLR	**N**
*OLR/basePool/device/***broadcast** • Each OLR **Sub** to this topic • Each OLR **Pub** on this topic to send messages to every other OLR.	**N**

Topics related to Races

Races currently in use in *the OLRNetwork*

OLRN Race Topic Root = **"OLR/basePool/race"**

Implemented: *(**N**)ot yet (**P**)artial (**Y**)es*

OLR/basePool/race/**status/<*RaceId*>** **(*) Races List** • **Each OLR Sub to** [*OLR/basePool/race/***status**/+*]* *to receive updates for the **Races list*** • The NetworkClient creating the race, or changing race status, **Pub** on the status/<Race*Id*> sub-topic	Y
*OLR/basePool/race/***<*RaceId*>/Participants/<*DeviceId*>** **(*) Race Participants List)** • On race "<RaceId>" creation, every client: ○ **Sub to** [.../<RaceId>/**Participants/**+] *to receive updates for the **Participants list*** • When a Client **Join a Race**=<RaceId>: ○ **Pub** on the <RaceId>/participants/<Device*Id*> sub-topic it's status	Y
*OLR/basePool/race/***<*RaceId*>/Config** • When a Client **Join a Race**=<RaceId>: ○ **Sub to** [*OLR/basePool/race/***<*RaceId*>/*Config***] *to receive **Config** params for the race (its order in the race, laps number, etc)* • The Client in charge of **Race Configuration**: ○ **Pub** on the <RaceId>/Config the complete Parameters Set	Y
*OLR/basePool/race/***<*RaceId*>/Cars/<CarId>** • When a Client **Join a Race**=<RaceId>: ○ **SUB to** [*OLR/basePool/race/***<*RaceId*>/*Cars*/+**] *to receive car's data (basically to know in wich OLR is the car in each moment)* • When a NetworkClient **Receive a Car** (car ENTER in the Device, coming from another): ○ **Pub** on the <RaceId>/cars/<CarId> to update the car's current OLR	Y
*OLR/basePool/race/***<*RaceId*>/Telemetry** Used in Race Visualization • The Devie currently active (i.e. with cars in it) **Pub** car's position data • A Race Visualization App will subscribe to this topic and display Race Sitution for each car	Y

OLR Network "Pool ID"

Is the "root string" for topics. As described above, every topic "starts" with:

- **OLR/<*PoolId*>/**

The "<***PoolId***>" string identifies a "SubSet" (group) of OLR Devices connected to the network.

User Interface will allow the user to choose a "Pool" (group) its device belongs to. Other devices using the same "**basePool**" (i.e in the same group) will be "visible" to make Relay Races.

*In the first implementation "PoolID" is not managed by UX - always set to: "**OLR/basePool/**" (Please note: This is managed in the "config.json" file. You can change it with no need to change the code)*

MQTT Payloads *- OLRNetwork messages*

We have seen the list of ‘Channels’ (topics) where the informations flows.
Now we’ll see the ‘Format” of the information transmitted in these channels.

>>> **One** Channel (MQTT topic) have **one** defined ‘message format’ <<<

JSON-encoded string is the preferred format used in messages (MQTT payloads).

Some channels, notably “*CarStatus*”, use a **plain text format**, to avoid JSON encode/decode overhead on send/receive

Message example: Payload for ***DeviceStatus* topic**

This was transmitted by a device with id=”TDO5e6cf279e3aed”

Sample message: Topic [OLR/AD2020/device/status/TDO5e6cf279e3aed]

```
{
  "VV": "0.4",                                   ← Protocol Version
  "TI": "TDO5e6cf279e3aed",                      ← Id
  "TM": "Harry.Tuttle",                          ← User Name
  "TN": "Test Track 3 – Naked LedStrip (IP30)",  ← Device Description
  "TS": "A1"                                     ← Status
}
```

*As you see, this channel uses a **JSON-encoded string.***

You may also see how JSON field names and some fields values are “encoded”:

(TS:A1 ***means*** “Status”:”Online”.)

In the following part of this doc you will find the definition of the **Specific format** of the message transmitted over every channel, plus the coded values used.

Device Status

*Topics: OLR/basePool/**device/status/**<DeviceId>*

Clients use this topic to **Publ**ish updates about their current status.

> *<DeviceId> indicates the ID of the Client Device publishing its status (**in other words each device have its own topic – no other device publish on it**)*

Message format for this channel: **JSON Encoded**

Sample MessageDeleted	
Topic:	OLR/basePool/device/status/TDO5e5f9d51ecc61
JSON Payload: {	
"VV": "0.4",	← **Protocol Version**
"TI": "TDO5e5f9d51ecc61",	← **Device Id**
"TM": "Harry.Tuttle",	← **User Name**
"TN": "HAM Test Track 1 - SLIM Case (left)",	← **Device Description**
"TS": "R0"	← **Status**
}	

(!) All fields in the message above are **required** *(any message sent on the channel **needs** to includes all of the fields above)*

Notes:

> The **DeviceId** field (**TI**) is redundant - is the "last part" of the topic.
> *Not a big overhead and the code results easier to understand.*

JSON Message Attributes	
VV	Protocol Version
TI	Device Id
TM	User who registered the Device
TN	Device Description
TS	Device Status → **Coded values** – *see below*

Device Status Attribute → Coded values	
00	Offline
01	Online - *Network Client register to the OLRNetwork*
A0	Available – *Successfull handshake with Device*
J1	Subscribing to Race
L0	Leaving a Race
R0	Subscribed to Race
R1	Configuring a Race
R2	Configuring Local Phisical Device
R3	Configured for Race
R9	Error configuring Phisical Device
R4	Ready to Start
R5	Racing
R6	Race Complete
R7	Play Race Again
GN	Not Responding

See source file: [Protocol.pde]→ *Class:* ***Protocol.Network.Channel.DeviceStatus***
Encode/Decode methods used when a message is Sent/Receives on the Network

Race Status

Topics: OLR/basePool/**race/status/<RaceId>**

Clients use this topic to send updates about Race Status.

*<**RaceId**> indicates the ID of the Race – Any devices participating to the Race may **Publ**ish on this topic*

Message format for this channel: **JSON Encoded**

Sample Message	
Topic:	OLR/basePool/race/status/**tIM7Yron7Qaz**

JSON Payload: {

"VV": "0.4",	**← Protocol Version**
"RS": "C1",	**← Race Status**
"RU": "TDO5e5f9d51ecc61",	**← Updating DeviceId (who publish the msg)**
"RI": "**tIM7Yron7Qaz**",	**← Race Id**
"RN": "HAM Test 2 Tracks"	**← Race Name**

}

(!) All fields in the message above are **required** *(any message sent on the channel **needs** to includes all of the fields above)*

Notes:

The RaceId field (**RI**) is redundant - is the "last part" of the topic.

Not a big overhead and the code results easier to understand.

JSON Message Attributes	
VV	Protocol Version
RI	Race Id
RN	Race Name
RU	Device Id of the Device who published the message (sender)
RS	Race Status → **Coded values** – *see below*

Race Status Attribute → Coded values	
A0	Acceping Participants
C1	Configuring
C2	Configured
C3	Waiting for Participant's Phisical Device Configuration
C4	Error configuring one of the OLR Participants
R0	Participans Configured
R3	Ready to Start
R4	Countdown
R5	Racing
R6	Paused
R7	Resumed
R8	Comlete
DD	Deleted

source file: [Protocol.pde] → *Class:* ***Protocol.Network.Channel.RaceStatus***
Encode/Decode methods used when a message is Sent/Receives on the Network.

Race Participants   *Topics: OLR/basePool/**race/<RaceId>/Participants/<DeviceId>***

A client use this topic to send updates about its “Status” as participant to the Race (Join, leave, etc).

*<**RaceId**> indicates the ID of the Race*
*<**DeviceId**> indicates the ID of the Client Device publishing its status as ‘Participant”.*
This means each device have its own topic – no other device publish on it

Message format for this channel: **JSON Encoded**

Sample Message	
Topic:	**OLR/basePool/**race/tlM7Yron7Qaz/Participants/TDO5e5f9d51ecc61
JSON Payload: { "VV": "0.4", "TI": "TDO5e5f9d51ecc61", "TM": "Harry.Tuttle", "TN": "HAM Test Track 1 - SLIM Case (left)", "TS": "L0" }	 ← **Protocol Version** ← **Device Id** ← **User Name** ← **Device Description** ← **Status**

In the example above, the Status=L0 means “Leaving the race”. *The message was sent by a Client “Leaving” a realy race after it finished.*

RaceParticipantStatus Channel uses **same message format as Device Status Channel**

– Please refer to Device Status Attributes and Coded Values

Race Configuration

Topics: *OLR/basePool/**race/<RaceId>/Config***

Used to share between participants the Configuration Parameters for a race (racetrack order, Laps, etc)

*<**RaceId**> indicates the ID of a specific*

- *Every participant **Sub** to the topic*
- *The Client in charge for configuration will **Pub** on the topic*

Message format for this channel: **JSON Encoded**

Sample Message	
Topic:	OLR/*basePool*/race/tIM7Yron7Qaz/Config

```
JSON Payload: {
  "VV": "0.4",      ← Protocol Version
  "RC": [           ← JSON ARRAY: One item for each Participant
   {                      ← Item #1: Cfg for First Participant
     "LO": 1,                   ← Laps
     "RE": 2,                   ← Repeat
     "TI": "TDO5e6bc02843177",← Device Id (Identify the Participant)
     "PO": 0                    ← Position
   },
   {                      ← Item #2: Cfg for Second Participant
     "LO": 1,
     "RE": 1,
     "TI": "TDO5e5f9d51ecc61",
     "PO": 1
   }
  ]
}
```

(!) The JSON Array will contain **one** Item **for each** participant.

In other words, the client in charge of the configuration will publish on this channel **one** message containing the configuration **for every participant**.

		JSON Message Attributes
VV	Protocol Version	
RC	JSON Array of Config Values for each Device	
	TI	Device Id
	LO	Laps for each section of the race in this device
	RE	How many times the Race passes through the circuit.
	PO	**Order** for the Relay Race. The device with lower Position (ex: 1) **will be the one where the race Starts**.

source file: [Protocol.pde] *→ Class: **Protocol.Network.Channel.RaceConfiguration***

Encode/Decode methods used when a message is Sent/Receives on the Network.

Car situation in Race OLR/basePool/**race/<RaceId>/Cars/<CarId>**

Messages in this topic are used, during a Race, to 'send' a car from one circuit to the next one.

*<**RaceId**> indicates the ID of the Race*
*<**CarId**> indicates the ID of the Car*

Participants **Sub** to [OLR/basePool/race/<RaceId>/Cars/+] to know where is the car *(in wich Racetrack is curently the Car)* .
When a Racetrack "*receives*" a Car (the car "*enter*" in the Device, coming from another), it **Pub**lish on this Channel to let everybody knows the car "*arrived*" in the Device.

Message format for this channel: **Plain Text**

This topic uses a plain text format, to avoid JSON encode/decode overhead on send/receive
CarStatus messages are exchanged to 'send' a car from one circuit to the next one. This process needs to be as fast as possible to minimize the 'lag' between "car leave" (the car disappears from Circuit A) and "car enter" (the car appears in Circuit B)

Sample Message	
Topic:	OLR/*basePool*/**race/tIM7Yron7Qaz/Cars/1**
Payload: "X:1,Red,TDO5e6bc02843177,3,8"	
The Payload is a string starting with a "Text Payload Header" (X:) and containing 5 comma-separated values: ***Car Id, CarName, CurrentDevice, Car Status, Car Speed***	
X:	TEXTPAYLOAD_HEADER Used in the protocol to identify the payload type=TEXT - *JSON payloads starts with "{"*
1	Car Id
Red	Car Name
TDO5e6bc02843177	Current Device Id (in which OLR Device the car is currently 'running'
3	**Car Status → Coded Values –see below**
8	Speed

Car Status → Coded values	
0	Stop
1	Racing
2	Leaving
3	Leaved
8	Winner

Race Telemetry OLR/basePool/**race/<RaceId>/Telemetry**

The device where a Car is currently ‘running’ send messages on this channel to describe the current car position:

> *<**RaceId**> indicates the ID of a specific Race*
> *Newtork Client software just **Publish** on this channel – Does not **Sub.***
> The data will be possibly used in the future to develop a “Race visualization App”

Message format for this channel: **JSON Encoded**

Sample Message	
Topic:	OLR/*basePool*/race/tIM7Yron7Qaz/Telemetry

```
JSON Payload: {
  "R": 66,                      ← Relative position of the Car in the Circuit
  "C": 2,                       ← Car Id
  "T": "M",                     ← Sub-Track Id
  "TI": "TDO5e5f9d51ecc61",     ← Device Id (Device where the Car is currently racing)
  "L": 1                        ← Lap Number
}
```

JSON Message Attributes	
TI	Device Id – OLR Device where the Car is currently racing.
C	Car Id
T	SubTrack Id – Section of the racetrack where the car is currently → **Coded Values – see below**
L	Current Lap Number ([1-99])
R	Relative Position in the track - *Expressed as a percentage ([00-99])*

SubTrack Id (T) → Coded values	
M	Stop
B	Racing
U	Leaving

Values in these fields comes directly from the Arduino Firmware.
*Please refer to the ‘**OLRN_Protocol_Serial**” doc for further details.*

Document revisions:

- 2020_04_24: Luca
  - Add: Message Attributes Tables
  - Doc cleanup
- 2019_09_15: Luca
  - First Publicly available version