MQTT PROTOCOL

DIGIRAIL OEE – V1.2x C

⚠	**Recommended for devices with firmware version V 1.2x and higher.**

1 INTRODUCTION

This document describes the required infrastructure, the data published by the device, and the operation mode of the **DigiRail OEE**, which can publish data to the cloud via the MQTT protocol. The device provides support for the following set of MQTT Brokers:

- Google IoT
- Microsoft Azure
- AWS
- NOVUS Cloud
- LiveMES
- Generic MQTT Broker (version 5.0 or higher)

2 PUBLISH AND SUBSCRIBE TOPICS

DigiRail OEE uses five topics. These topics are defined in the device configuration process and stored in the following variables:

- **Device data:** Used to publish the data generated in the device. It has two types: **`channel`** and **`events`**.
- **Config:** Used to send configuration data to the device. The device subscribes to this topic and shows the updates in the **Config Ack** topic.
- **Config Ack:** The device publishes the current configuration in this topic. If the **Config** topic receives a new configuration, this topic will confirm whether the new configuration has been applied.
- **Command:** The device receives (subscribe) commands through this topic. The result of this command is published in the **Command Ack** topic.
- **Command Ack:** The device publishes the result of commands executed in this topic.

See the table below:

TOPIC	PUB/SUB	USE
Device data	*Publish*	Publishes the data generated by the device. This topic receives Channel Data and Events.
Config	*Subscribe*	Receives the configuration data.
Config Ack	*Publish*	Responds to the configuration data.
Command	*Subscribe*	Receives commands. This topic receives Output, Reset counters, RS485 MQT Gateway and Get diagnostic commands.
Command Ack	*Publish*	Responds to the execution of the commands.

Table 1 – Publish and subscribe topics

2.1 DATA AND EVENT PUBLISHING MODEL

Publishing of the events and data generated by the device follows the standard MQTT template and uses a topic defined during configuration.

2.2 CONFIGURATION AND COMMAND PUBLISHING MODEL

The basic model of how the commands and settings work was based on the device twins implementation of the Microsoft Azure cloud, which, as described in Understand and use device twins in IoT Hub, is used to synchronize device settings and conditions.

This model has two basic concepts:

- **`Desired properties`:** These are the conditions and settings that the backend application can change or query on the device it is interacting with.
- **`Reported properties`:** These are used as a response after receiving Desired properties. The device reports the current status or the result of a command.

This message exchange model needs two different topics to work. The first is the topic in which the device is subscribed to receive **`Desired properties`**. This step, initiated by the application, is called "**`request`**". In the second topic, the device will publish the **`Reported properties`** after executing a command or applying a configuration. This step is called "**`response`**".

3 DATA AND EVENTS

The data will be published to the topic defined in the variable **`Device data`**. The data type is indicated in the JSON of the message. You should note the following items for all data:

3.1 CHANNEL DATA

The channel data is published periodically, according to the device configuration. The data is in JSON format and has the following key/value sets:

```
{
    "device_id": "device0",
    "channels" : {
        "timestamp":1585819219,
        "chd1_value":0,
        "chd2_value":0,
        "chd3_value":0,
        "chd4_value":0,
        "chd5_value":0,
        "chd6_value":0,
        "ch1_user_range":2.17,
        "ch2_user_range":2.2
    }
}
```

Notes:

- **`timestamp`** is in UTC.

3.2 EVENTS

The event data will be published whenever a previously configured event in the device occurs. The data is in JSON format and has the following key/value sets:

```
{
    "device_id": "device0",
    "events": {
        "chd1": {
            "timestamp":1585819219.685,
            "edge":1,
        }
    }
}
```

Notes:

- The **`timestamp`** value is also in UTC, but in double format, with the milliseconds of the event in the fractional part.

4 CONFIGURATIONS

You can change or query the device settings by publishing to the topic defined in the **Config** variable. In the **Config Ack** topic you can see if the changes have been executed and query their current status.

The configuration items for this type of device are as follows:

CONFIGURATION ITEMS	DESCRIPTION
rtc	Setting of the RTC (Real Time Clock).
device	General device configuration.
chdX	Configuration of the digital channel 'X' (Available: **chd1**, **chd2**, **chd3**, **chd4**, **chd5** and **chd6**).
periodic counter reset	Configuration of the reset periodicity of the digital counters.
chX	Configuration of the analog channel 'X' (Available: **ch1** and **ch2**).
eth	Ethernet interface configuration (when available).
wifi	Wi-Fi interface configuration (if available).
modbus tcp	Modbus-TCP protocol configuration.
rs485	RS485 interface configuration.

Table 2 – Configuration items

4.1 HOW TO CHANGE THE CONFIGURATION PARAMETERS

The steps to change the configuration are as follows:

STEP	ACTION
1	Send the configuration **request** when publishing to the **Config** topic.
2	The device, which is subscribed to the **Config** topic, evaluates, and applies the new configuration.
3	The device publishes the response in the **Config Ack** topic.
4	The application, which is subscribed to the **Config Ack** topic, updates the device status and the result of the operation according to what it received in the response message.

Table 3 – Steps

The data that was used when sending and receiving the configuration is in JSON format and is present in the payload of the messages that were exchanged between the application and the device.

The structure of the configuration **request** received by the device is as follows:

```
{
    "timestamp":1585819219,
    "desired": {
        <desired item>
    }
}
```

The value given in **timestamp** is in UTC and serves to identify the configuration **request** message. The corresponding **response** will have the same value as the one received.

A **request** can contain only one item to be configured, called <**desired item**>, and you can send only the key/value pairs you want to change, omitting the others.

At the end of the execution, the device sends the response in the **Config Ack** topic, reporting the result of the operation for each configured item in the following format:

```
{
    "device_id": "device0",
    "timestamp":1585819219,
    "reported": {
        <reported item>
    }
}
```

The **timestamp** value in the **response** message is the same as in the configuration **request** message.

The configuration **request** publication can have only one **desired item** for each message. In most cases, the data structure of the **reported item** is the same as that of the **desired item**, but with the addition of an **item error** that indicates the result in the application of the respective **desired item**. Exceptions are indicated in each of the configuration items below.

4.2 HOW TO HANDLE ERRORS WHEN CHANGING THE CONFIGURATION

The values set in each **`desired item`** of the **`request`** will only be applied if the execution has no errors in any of the key/value pairs sent in that **`desired item`**. Each **`desired item`** is processed independently. There can be different response messages for each **`response item`**.

The **`error`** value is an integer and reports the first error encountered when applying the configuration for an item.

The table below shows the error codes:

CODE	DESCRIPTION
0	Success.
1	The structure is correct, but the device has received a parameter that is out of range.
2	The structure is correct, but the device received an unknown parameter.

Table 4 – Error codes

The table below shows the device actions for each error condition:

ERROR CONDITION	ACTION
Configuration item not recognized	The **`reported item`** will only contain the error value in **`error`**.
Error when applying a configuration	The **`reported item`** will contain the current configuration values and the error value will indicate the first error that occurred.

Table 5 – Device actions

4.3 HOW TO CONSULT THE CONFIGURATION PARAMETERS

The steps for querying the current configuration are as follows:

STEPS	ACTION
1	Send the **`request`**, indicating the configuration item that is to be queried in the **Config** topic.
2	The device, which is subscribed to the **Config** topic, evaluates the request, and reads the configuration data.
3	The device publishes the response in the **Config Ack** topic.
4	The application, which is subscribed to the **Config Ack** topic, updates the device status according to the data in the response message.

Table 6 – Configuration query steps

The data that was used when sending and receiving the configuration is in JSON format and is present in the payload of the messages that were exchanged between the application and the device.

The structure of the configuration **`request`** received by the device is as follows:

```
{
    "timestamp":1585819219,
    "desired": {
        <empty desired item>
    }
}
```

The **`timestamp`** follows the default used when changing the configuration. As shown the above example, a **`request`** can contain only one item to be queried, called an <**`empty desired item`**>.

An **`empty desired item`** is a configuration item with no key/value pairs, as shown in the example below:

```
{
    "timestamp":1585819219,
    "desired": {
        "rtc" : {}
    }
}
```

In the example above, the corresponding **`response`** will have the value of the current RTC.

The configuration **`request`** publication can have multiple **`empty desired items`**, one for each item you want to query. The data structure of the **`reported items`** is the same as that used in the **`response`** you receive when changing parameters. If the queried item exists, the **`error`** value will indicate that the operation was successful.

4.4 HOW TO HANDLE ERRORS WHEN QUERYING THE CONFIGURATION

Each **`empty desired item`** is processed independently. The configuration **`response`** messages can have different return status. The **`error`** value is an integer and reports the first error encountered when reading the configuration of an item.

The table below shows the device actions for each error condition:

ERROR CONDITION	ACTION
Configuration item not recognized	The **`reported item`** will only contain the error value in **`error`**.
Error when applying a configuration	The **`reported item`** will contain the current configuration values and the error value will indicate the first error that occurred.

Table 7 – Device actions

4.5 CONFIGURATION ITEMS

4.5.1 RTC

REQUEST *RTC*

```
{
    "timestamp":1585819219,
    "desired": {
        "rtc": {
            "year":2021,
            "month":2,
            "day":25,
            "hour":12,
            "minute":13,
            "sec":10
        }
    }
}
```

RESPONSE *RTC*

```
{
    "device_id": "device0",
    "timestamp":1585819219,
    "reported": {
        "rtc": {
            "error": 0,
            "year":2021,
            "month":2,
            "day":25,
            "hour":12,
            "minute":13,
            "sec":10
        }
    }
}
```

4.5.2 DEVICE

REQUEST *DEVICE*

```
{
    "timestamp":1585819219,
    "desired": {
        "device": {
            "title":"Pci",
            "location":"location_123",
            "pub_interval":60,
           "alter_pub_interval_enable":1,
           "alter_pub_interval":600,
        }
    }
}
```

RESPONSE *DEVICE*

```
{
    "device_id": "device0",
    "timestamp":1585819219,
    "reported": {
        "device": {
            "error": 0,
            "title":"Pci",
            "location":"location_123",
            "pub_interval":60,
        }
    }
}
```

4.5.3 DIGITAL CHANNELS

The example in this section shows only digital channel 1, indicated as **chd1**. The other channels (**chd2**, **chd3**, **chd4**, **chd5** and **chd6**) follow the same data model.

REQUEST DIGITAL CHANNELS

```
{
    "timestamp":1585819219,
    "desired": {
        "chd1": {
            "enable":1,
            "counting_m":2,
            "type":3,
            "edge":1,
            "debounce":555,
            "reset_m":2,
                "debounce_enable":0
                 }
            }
        }
```

RESPONSE DIGITAL CHANNELS

```
{
    "device_id": "device0",
    "timestamp":1585819219,
    "reported": {
        "chd1": {
            "error": 0,
            "enable":1,
            "counting_m":2,
            "type":3,
            "edge":1,
            "debounce":555,
            "reset_m":2
                "debounce_enable":0
                 }
            }
        }
```

4.5.4 PERIODIC COUNTER RESET

REQUEST CHD_PERIODIC_RESET

```
{
    "timestamp":1585819219,
    "desired": {
        "chd_periodic_reset" : {
            "type":0,
            "day":2,
            "hour":3,
            "minute":4,
            "sec":5,
            "week_day":6
        }
    }
}
```

RESPONSE CHD_PERIODIC_RESET

```
{
    "device_id": "device0",
    "timestamp":1585819219,
    "reported": {
        "chd_periodic_reset" : {
            "error": 0,
            "type":0,
            "day":2,
            "hour":3,
            "minute":4,
            "sec":5,
            "week_day":6
        }
    }
}
```

4.5.5 ANALOG CHANNELS

The example in this section shows only analog channel 1, indicated as **`ch1`**. The other channel (**`chd2`**) follows the same data model.

REQUEST ANALOG CHANNELS

```
{
    "timestamp":1585819219,
    "desired": {
        "ch1" : {
            "enable":1,
            "sensor_type":1,
            "range_min":-10,
            "range_max":2020,
            "decimal_point":2
       }
   }
}
```

RESPONSE ANALOG CHANNELS

```
{
    "device_id": "device0",
    "timestamp":1585819219,
    "reported": {
        "ch1" : {
            "error": 0,
            "enable":1,
            "sensor_type":1,
            "range_min":-10,
            "range_max":2020,
            "decimal_point":2
        }
    }
}
```

4.5.6 ETHERNET

REQUEST *ETHERNET*

```
{
    "timestamp":1585819219,
    "desired": {
        "eth" : {
            "enable_dhcp":0,
            "addr":[10, 167, 2, 3],
            "mask":[255,255, 255, 0],
            "gateway":[255, 255, 255, 0],
            "ipv4dns":[8, 8, 8, 8]
        }
    }
}
```

RESPONSE *ETHERNET*

```
{
    "device_id": "device0",
    "timestamp":1585819219,
    "reported": {
        "eth" : {
            "error": 0,
            "enable_dhcp":0,
            "addr":[10, 167, 2, 3],
            "mask":[255,255, 255, 0],
            "gateway":[255, 255, 255, 0],
            "ipv4dns":[8, 8, 8, 8]
        }
    }
}
```

4.5.7 WI-FI

REQUEST *WIFI*

```
{
    "timestamp":1585819219,
    "desired": {
        "wifi" : {
            "ssid":"WifiName",
            "pwd":"password"
        }
    }
}
```

RESPONSE *WIFI*

```
{
    "device_id": "device0",
    "timestamp":1585819219,
    "reported": {
        "wifi" : {
            "error": 0,
            "ssid":"WifiName"
        }
    }
}
```

Note:

- The **pwd** key is not transmitted in the response.

4.5.8 MODBUS-TCP

REQUEST MODBUS TCP

```
{
    "timestamp":1585819219,
    "desired": {
        "modbus_tcp" : {
            "enable":1,
            "port":502
        }
    }
}
```

RESPONSE MODBUS TCP

```
{
    "device_id": "device0",
    "timestamp":1585819219,
    " reported ": {
        "modbus_tcp" : {
            "error": 0,
            "enable":1,
            "port":502
        }
    }
}
```

4.5.9 RS485

REQUEST *RS 485*

```
{
    "timestamp":1585819219,
    "desired": {
        "rs485" : {
            "baudrate":6,
            "stopbits":1,
            "parity":1,
            "timeout":500
        }
    }
}
```

Note:

- The **`timeout`** key has a value in milliseconds.

RESPONSE *RS 485*

```
{
    "device_id": "device0",
    "timestamp":1585819219,
    "reported": {
        "rs485" : {
            "error": 0,
            "baudrate":6,
            "stopbits":1,
            "parity":1,
            "timeout":500
        }
    }
}
```

5 COMMANDS

The data will be published to the topic defined in the variable **Command**. The data type is indicated in the JSON of the message. The return from the execution of the commands occurs through the **Command Ack** topic.

5.1 OUTPUT

This command changes the status of the device outputs.

5.1.1 REQUEST OUTPUT

```
{
    "timestamp":1585819219,
    "desired": {
        "output" : {
            "out1":1,
            "out2":1
        }
    }
}
```

Note:

- Status that will not be modified do not need to be published.

5.1.2 RESPONSE OUTPUT

```
{
    "device_id": "device0",
    "timestamp":1585819219,
    "reported": {
        "output" : {
            "error": 0,
            "out1":1,
            "out2":1
        }
    }
}
```

Notes:

- The **`timestamp`** is the same as the received command.
- The status described in **`desired`** will only be applied if execution is done without errors.
- The value of **`error`** is an integer and reports the first error found during the execution of the command.
- If the command failed, the status reported will be the current.

5.2 RESET COUNTERS

This command resets the digital channel counters.

5.2.1 REQUEST RESET COUNTERS

```
{
    "timestamp":1585819219,
    "desired": {
        "reset_counters" : {
            "reset_chd2":1,
            "reset_chd4":1,
        }
    }
}
```

Note:

- Counters that will not be reset do not need to be published

5.2.2 RESPONSE RESET COUNTERS

```
{
    "device_id": "device0",
    "timestamp":1585819219,
    "reported" : {
        "reset_counters": {
           "error": 0,
           "reset_chd1":0,
           "reset_chd2":0,
           "reset_chd3":0,
           "reset_chd4":0,
            "reset_chd5":0,
            "reset_chd6":0
        }
    }
}
```

Notes:

- The **`timestamp`** is the same as the received command.
- The status described in **`desired`** will only be applied if execution is done without errors.
- The value of **`error`** is an integer and reports the first error found during the execution of the command.
- The **`reset_chdX`** keys (with X from 1 to 6) can assume values 0 or 1. When the value is 1, the counter will be reset. The value 0 indicates that the counter should not be changed.

5.3 SET COUNTERS

This command changes the value of the digital channels counters.

5.3.1 REQUEST SET COUNTERS

```
{
    "timestamp":1620413979
    "desired": {
      "set_counters" : {
               "set_chd2":6500,
               "set_chd3":10
          }
    }
}
```

Nota:

Counters that will not be changed should not be published.

5.3.2 RESPONSE SET COUNTERS

```
{
    "device_id": "device0",
    "timestamp":1620413979,
    "reported" : {
        "set_counters": {
           "error": 0,
           "set_chd1":0,
           "set_chd2":6500,
           "set_chd3":10,
           "set_chd4":0,
           "set_chd5":0,
           "set_chd6":0
        }
    }
}
```

Notes:

- The **`timestamp`** is the same as the command received (**`desired`**).
- The status described in the **`desired`** step will only be applied if the execution is done without errors.
- The **`error`** value is an integer and reports the error found during the command execution.
- In this example, digital channels 1, 4, 5 and 6 do not appear in the JSON **`desired`** since you do not want to change their counters. In the response, the current value of the digital channel will be returned. For digital channels 1, 4, 5 and 6 the current value is assumed to be zero.

5.4 GATEWAY MQTT RS485

This command sends the bytes of mb_buffer over the RS485 interface. The value of each byte contained in mb_buffer must be in hexadecimal format.

5.4.1 REQUEST GATEWAY MQTT RS485

```
{
    "timestamp":15,
    "desired": {
        "gateway_485": {
            "mb_buffer":"02 03 00 00 00 0A C5 FE"
         }
    }
}
```

Notes:

- In the **`response`** step of the MQTT command, the bytes received on the RS485 interface are contained in mb_buffer.
- If the device timeout is addressed to the RS485 interface, mb_buffer will return empty.

5.4.2 RESPONSE GATEWAY MQTT RS485

```
{
    "device_id":"DeviceName",
    "timestamp":15,
    "reported": {
            "gateway_485": {
                "error":0,
                "mb_buffer":"02 03 14 19 C7 00 00 06 4E 00 00 04 E0 00 00 03 D0 00 00 03 D0 00 00
              1B 13"
        }
    }
}
```

Notes:

- The **`timestamp`** is the same as the received command.
- The value of **`error`** is an integer and reports the error found during the execution of the command.

5.5 GET DIAGNOSTIC

5.5.1 REQUEST GET DIAGNOSTIC

```
{
    "timestamp":1585819219,
    "desired" : {
        "diag" : {}
    }
}
```

5.5.2 RESPONSE GET DIAGNOSTIC

```
{
    "device_id": "device0",
    "timestamp":1585819219,
    "reported" : {
       "diag": {
            "title": "Pci v2",
            "location":"home",
            "curr_timestamp":1589326517,
            "cfg_timestamp":1589311676,
            "fw_v":"01.00",
            "mqtt_queue":1,
            "sn":"00000001",
            "curr_rssi":"55",
            "min_rssi":"46",
            "max_rssi":"87",
            "avg_rssi":"54",
            "ipv4":[ 192, 168, 0, 23 ]
        }
    }
}
```

If the **Publish Diagnostics Periodically** parameter of the **NXperience** configuration software (see the MQTT Protocol section of the Configuration Software chapter in the **DigiRail OEE** manual) is enabled, the occurrence counters for the system events will also be added to the response:

```
{
  "device_id": "droee",
  "timestamp": 1585819219,
  "reported": {
    "diag": {
      "error": 0,
      "title": "Pci v2",
      "location": " home ",
      "curr_timestamp": 1589326517,
      "cfg_timestamp": 1589311676,
      "fw_v": "1.23",
      "mqtt_queue": 1,
      "sn": "00000001",
      "curr_rssi": "55",
      "min_rssi": "45",
      "max_rssi": "70",
      "avg_rssi": "55",
      "ipv4": [ 192, 168, 0, 23 ],
      "log_counters": {
        "pwr_on": 1,
        "pwr_sw_reset": 0,
        "net_disconnected": 1,
        "wifi_prov_error": 0,
        "dhcp_error": 0,
        "dns_error_1": 0,
        "dns_error_2": 0,
        "cfg_updated": 1,
        "fw_updated": 0
      },
      "watchdog_counters": {
        "analog": "0",
        "data_storage": "0",
        "record_storage": "0",
        "digital": "0",
        "modbus": "0",
        "record_periodic": "0",
        "mqtt": "1",
        "network": "0"
      }
    }
  }
}
```

5.6 RESET DIAGNOSTIC

The **reset diag** command is used so that the application can reset counters related to internal system events and Wi-Fi signal quality (RSSI) measurement data.

The values of the **reset_watchdog_counter**, **reset_x_counter** and **reset_diag_rssi** fields can have values of 0 or 1. The value "1" means that a reset is to be applied to the corresponding parameter. The value "0" indicates that the parameter should not be changed. In this case you can also simply omit the JSON channel.

5.6.1 REQUEST RESET DIAGNOSTIC

```
{
    "timestamp":1585819219,
    "desired": {
        "reset_diag": {
            "reset_watchdog_counter":0,
            "reset_logs_counter":1,
            "reset_diag_rssi":1
        }
    }
}
```

5.6.2 RESPONSE RESET DIAGNOSTIC

```
{
    "device_id": "device0",
    "timestamp":1585819219,
    "reported": {
        "reset_diag": {
           "error": 0,
           "reset_watchdog_counter":0,
           "reset_logs_counter":0,
            "reset_diag_rssi":0
        }
    }
}
```

Notes:

- The **`timestamp`** is the same as the received command (**`desired`**).
- The status described in **`desired`** will only be applied if execution is done without errors.
- The value of **`error`** is an integer and reports the error encountered during the execution of the command.

5.7 LOGS

The logs command returns the last 50 log events from the system. All events will have an ID, which can be queried with this command, and a timestamp of the occurrence. You can see a detailed description of the log in **Table 8**.

5.7.1 REQUEST LOGS

```
{
    "timestamp":1585819219,
    "desired": {
        "logs": {}
    }
}
```

5.7.2 RESPONSE LOGS

```
{
  "device_id": "droee",
  "timestamp": 1585819219,
  "reported": {
    "logs": {
      "error": 0,
      "events": [ {
        "ts": 1638193059,
        "id": 9
      }, {
        "ts": 1638193055,
        "id": 10
      }, {
        "ts": 1638192333,
        "id": 9
      }, {
        "ts": 1636466491,
        "id": 4
      } ]
    }
  }
}
```

5.8 LOGS_PARSED

Due to device memory limitations, the **`logs_parsed`** command returns only the last 30 system log events. However, instead of giving an ID, there will be a short description of the log, plus the timestamp of the occurrence, like the **`logs`** command. You can see a detailed description of the log in **Table 8**.

5.8.1 REQUEST LOGS_PARSED

```
{
    “timestamp”:1585819219,
    “desired”: {
        “logs_parsed”: {}
    }
}
```

5.8.2 RESPONSE LOGS_PARSED

```
{
  "device_id": "droee",
  "timestamp": 1585819219,
  "reported": {
    "logs_parsed": {
      "error": 0,
      "events": [ {
        "ts": 1638193059,
        "mqtt": "connected"
      }, {
        "ts": 1638193055,
        "mqtt": "disconnected"
      }, {
        "ts": 1638192333,
        "mqtt": "connected"
      }, {
        "ts": 1636468024,
        "net": "connected"
      } ]
    }
  }
}
```

The table below shows a detailed description of the logs:

CODES	LOGS_PARSED		DESCRIPTION
0	pwr	on	Standard startup.
1	pwr	sw_reset	Startup triggered by software reset.
2	pwr	wdt_reset	Startup triggered by internal Watchdog.
3	pwr	lvd_reset	Startup triggered by power outage.
4	net	connected	Connected to a network (Wi-Fi or Ethernet).
5	net	disconnected	Disconnected from the network (Wi-Fi or Ethernet).
6	wifi	prov_error	Wi-Fi provisioning failure (SSID or password incorrect).
7	dhcp	error	DHCP error.
8	sntp	error	SNTP error.
9	mqtt	connected	Connected to a MQTT broker.
10	mqtt	disconnected	Disconnected from the MQTT broker.
11	mqtt	sub_error	MQTT topics subscription error.
12	mqtt	pub_error	MQTT topics publishing error.
13	mqtt	alter_int	Publish interval has been changed to an alternative interval.
14	mqtt	default_int	Publish interval has been changed to a default interval.
15	dns	error_1	DNS internal error - Phase 1.
16	dns	error_2	DNS internal error - Phase 2.

CODES	LOGS_PARSED		DESCRIPTION
17	dns	error_3	DNS internal error - Phase 3.
18	mem	init_error	There was an error during the circular buffer initialization. Device has recovered.
19	mem	not_init	The circular buffer has not been initialized.
20	mem	read_error	There was a failure while reading the circular buffer.
21	cfg	updated	Device configuration updated.
22	fw	updated	Device firmware updated.

Table 8 – Codes

6 TOPICS IN MULTIPLE CLOUDS

The topics used by the device will be configured according to the cloud type you select. Topics are unique to one device, which is identified by the **`{id}`** variable. This variable is supplied during the configuration process.

6.1 AWS

VARIABLE	TOPIC
Device data	NOVUS/{id}/events
Config	NOVUS/{id}/config
Config Ack	NOVUS/{id}/ack/config
Command	NOVUS/{id}/command
Command Ack	NOVUS/{id}/ack/command

Table 9 – AWS

6.2 GOOGLE IOT

VARIABLE	TOPIC
Device data	/devices/{id}/events
Config	/devices/{id}/commands/#
Config Ack	/devices/{id}/events
Command	/devices/{id}/commands/#
Command Ack	/devices/{id}/events

Table 10 – Google IoT

6.3 MICROSOFT AZURE

VARIABLE	TOPIC
Device data	devices/{id}/events/
Config	devices/{id}/messages/devicebound/#
Config Ack	devices/{id}/events/
Command	devices/{id}/messages/devicebound/#
Command Ack	devices/{id}/events/

Table 11 – Microsoft Azure

6.4 NOVUS CLOUD

VARIABLE	TOPIC
Device data	NOVUS/{id}/events
Config	NOVUS/{id}/config
Config Ack	NOVUS/{id}/ack/config
Command	NOVUS/{id}/command
Command Ack	NOVUS/{id}/ack/command

Table 12 – NOVUS Cloud

6.5 LIVEMES

VARIABLE	TOPIC
Data PUB	devices/novus/doee/{ID_Dispositivo}/data
Config Ack Pub	devices/novus/doee/{ID_Dispositivo}/config-ack
Command Ack PUB	devices/novus/doee/{ID_Dispositivo}/command-ack
Config SUB	devices/novus/doee/{ID_Dispositivo}/config/#
Command SUB	devices/novus/doee/{ID_Dispositivo}/command/#

Table 13 – LiveMES

6.6 MINA

VARIABLE	TOPIC
Data PUB	devices/novus/doee/{ID_Dispositivo}/data
Config Ack Pub	devices/novus/doee/{ID_Dispositivo}/ack/config
Ack PUB Command	devices/novus/doee/{ID_Dispositivo}/ack/command
Config SUB	devices/novus/doee/{ID_Dispositivo}/config
Command SUB	devices/novus/doee/{ID_Dispositivo}/command

Table 14 – Mina

6.7 GENERIC BROKER

The parameters for a Broker that has been defined by the user can be set arbitrarily. To improve device performance, it is recommended to use the **Device data**, **Config Ack**, and **Command Ack** topics for publishing the device, and the **Config** and **Command** topics for publishing the device management/configuration application.

VARIABLE	TOPIC
Device data	Device Publish topic
Config	Device Subscribe topic
Config Ack	Device Publish topic
Command	Device Subscribe topic
Command Ack	Device Publish topic

Table 15 – Generic Broker

7 CONFIGURATION VARIABLES

These are the configuration variables allowed by the MQTT protocol:

CORRESPONDING CONFIGURATION ITEM	VARIABLE	DESCRIPTION	MINIMUM VALUE	MAXIMUM VALUE
RTC	year	Allows you to configure the year to be used to set the device RTC.	2016	2080
	month	Allows you to configure the month to be used to set the device RTC.	1	12
	day	Allows you to configure the day to be used to set the device RTC.	1	28
	hour	Allows you to configure the hour to be used to set the device RTC.	0	23
	minute	Allows you to configure the minute to be used to set the device RTC.	0	59
	sec	Allows you to configure the second to be used to set the device RTC.	0	59
DEVICE	title	Allows you to define a device name.	-	20
	location	Allows you to inform where the device has been positioned.	-	40
	pub_interval	Allows you to configure the interval at which the data will be published to the MQTT Broker (in seconds).	1	65535
	alter_pub_interval_enable	Allows you to enable an alternative publishing interval whenever there are connection problems with the Broker MQTT.	0	1
	alter_pub_interval	Allows you to configure an alternative publishing interval whenever there are connection problems with the Broker MQTT.	60	65535
DIGITAL CHANNELS	enable	Allows you to enable the digital channel.	0	1
	counting_m	Allows you to configure the counting mode for the digital channel: 0 → Not defined 1 → Counter 2 → Event	0	2
	type	Allows you to configure the sensor type of the digital channel: 0 → Not configured 1 → PNP 2 → NPN 3 → Dry contact	0	3
	edge	Allows you to configure the counting edge of digital channel: 1 → Rising edge 2 → Falling edge 3 → Both edges	1	3
	debounce	Allows you to configure the digital channel Debounce time for the Dry Contact sensor type (in milliseconds).	0	60000
	reset_m	Allows you to configure the reset mode of the digital channel accumulators: Bit 0 → Overflow Bit 1 → Calendar Bit 2 → Protocol	0	2
	debounce_enable	Allows you to enable Debounce for the digital channel.	0	1
PERIODIC COUNTERS RESET	type	Allows you to configure the digital counters reset mode: 0 → Daily 1 → Weekly 2 → Monthly	0	2
	day	Allows you to configure the reset day for the counter.	0	28
	hour	Allows you to configure the reset hour for the counter.	0	23
	minute	Allows you to configure the reset minute for the counter.	0	59
	sec	Allows you to configure the reset second for the counter.	0	59
	week_day	Allows you to configure the reset week for the counter.	1	7
ANALOG CHANNELS	enable	Allows you to enable the analog channel.	0	1
	sensor_type	Allows you to configure the sensor type of the analog channel: 0 → Not defined 0 → 0-5 V 2 → 0-10 V	0	4

CORRESPONDING CONFIGURATION ITEM	VARIABLE	DESCRIPTION	MINIMUM VALUE	MAXIMUM VALUE
		3 → 0-20 mA 4 → 4-20 mA		
	range_min	Allows you to configure the analog channel minimum limit.	0	0xFFFF
	range_max	Allows you to configure the analog channel maximum limit.	0	0xFFFF
	decimal_point	Allows you to configure the decimal place of the analog channel (fixed point for display and memory register): 0 → No decimal places 1 → One decimal place 2 → Two decimal places	0	2
ETHERNET	enable_dhcp	Allows you to define that the device gets its IP via DHCP.	0	1
	addr	Allows you to configure the device IPv4 address.	0	65535
	mask	Allows you to configure the network mask.	0	65535
	gateway	Allows you to configure the network Gateway.	0	65535
	ipv4dns	Allows you to configure the DNS server IP.	0	65535
WI-FI	ssid	Allows you to configure the Wi-Fi network SSID.	0x0000	0xFFFF
	password	Allows you to configure a Wi-Fi network password.	0x0000	0xFFFF
MODBUS-TCP	enable	Allows you to enable the Modbus-TCP protocol.	0	1
	port	Allows you to configure a communication port for the Modbus-TCP protocol.	0	0xFFFF
RS485	baudrate	Allows you to configure the RS485 interface Baud Rate: 0 → 1200 1 → 2400 2 → 4800 3 → 9600 4 → 19200 5 → 38400 6 → 57600 7 → 115200	0	7
	stopbits	Allows you to configure the RS485 interface Stop Bits: 0 → 1 Stop Bit 1 → 2 Stop Bits	0	1
	parity	Allows you to configure the RS485 interface parity: 0 → No parity 1 → Odd parity 2 → Even parity	0	2
	timeout	Allows you to configure a timeout value for the connection (in milliseconds).	0	65535

Table 16 – Configuration Variables Table