Alleantia
ALL <THINGS> JOINED

USER MANUAL
MQTT PROTOCOL


Alleantia
ALL <THINGS> JOINED

Indice

Introduction

1. Plant configuration

The gateway publishes the data of the devices, monitored on the customer's MQTT broker.

Once the Alleantia software has been set up in the field, the broker addressing and the frequency of sending data are entered, the system will automatically send them.

It will also be necessary to select the variables, alarms and events beforehand to be forwarded through the MQTT service.

2. Configuration and initial mapping

First, the configuration and mapping of the system must be scanned, in order to recreate its state:
- the number of gateways in the field: each of them will have its own Serial Id.

For each gateway in the field, first query phase is necessary to create a lookup table to keep the data decryption keys.

These queries are reported in Section 3.2 onwards in this manual, in order to understand and to map:

- the number of devices in the field and their unique nomenclature (dev Id): this is the number of equipment/ machines monitored by the specific software license;

- the number of variables that underlie each device (var id): these variables are the data such as speed, temperature, etc., transmitted by the machine.

Once the mapping of the system has been rebuilt, only the receipt of the value of the variable (value) is needed, which is communicated automatically by the device in the Telemetry Topic.

3. Topics

The data published by the system are divided into Topics.
Each gateway will publish messages on 3 different Topics:
- Topic Alarms
- Topic Events
- Topic Telemetry

The initial configuration has a fourth Topic, which the gateway listens to and waits for requests from the outside, according to the syntax (see Section 3):
- Topic Commands.

Each Topic has the following pattern: Serial Id/Telemetry, Serial Id/Alarm, Serial Id/Events, Serial Id/Commands.

Therefore, the Topic Commands will be used in order to realize the initial requests from the user, to rebuild the lookup of the system.

All the responses from the gateway will take place on the publication Topics mentioned above, especially the Topic Telemetry.

Introduction

Only information related to alarms or events, set in the system (Alleantia software) will be published on the related Topic Alarms and Events.

Note:
Every addition in the field of monitored devices and variation of the configuration implies updating the lookup and the replication of requests for information to the Topic Commands.

Final summary

First, make connection to the gateways in the field in order to determine which machines are monitored and which variables each machine communicates.

Following the mapping of the system it will be possible to read the values of the variables whose origin (machine and gateway) will be known.

There is a set of initial calls on the Topic Commands and the gateway will respond on the Topic Telemetry. The automated information on 3 Publication Topics will be sent regularly.

1 MQTT Service Description

IoT Scada provides the message forwarding service to the cloud, using MQTT protocol.

If the license is enabled and if the serivce has been activated, IoT Scada will regularly send the telemetry messages as a string in JSON format to configured MQTT broker.

The IoT Scada provides a mechanism for configuring permissions (read/write/ forward alarms) of the variables to forward, therefore only the variables with such permissions will be forwarded through the telemetry messages. Even the writing operation (if logically possible) will be carried out only if such permission has been previously enabled in the configuration.

There are three types of messagesthat can be forwarded automatically from the devices, wich differ in the amount of provided information and the syntax (and therefore in messages size).

In the IoT Scada, every single variable, collected from the field, is uniquely identified by the **devId** (device identifier the variable belongs to) and **varId (variable ide ntifier).** Some variables may have an unadjiusted **devId:** these are so-called **Plant variables** (called Custom Measures in the IoT Scada), that do not have such identifier due to their nature: they do not belong to any device, but they are the result of an elaborationmade on several variables, basically coming from different devices, (example: he sum of KWh read by N inverters connected to the IoT Scada).

The operating principle ensures that only the minimum amount of data is forwarded provides that a first, only one query is performed during inizialization, by forwarding one or more messages in JSON format (explaining later in this manual) to the broker, to get as a response the device configuration, connected devices, configuration of variables, alarms, events, and to build a lookup table with them.

In this way, the user will have the keys to identify the data forwarded automatically in the subsequent telemetry messages, so that this information should not be forwarded again in every message.

2 Telemetry messages

The telemetry messages, automatically forwarded with the frequency, that can be set in the device interface, have the following syntax:

2.1 Complete message

```
{
  “telemetryDataList”:[
    {
      “date”:”Jan 8, 2018 10:50:30 AM”, //java.util.Date
      “category”:”main”, //String
      “blockId”:”B2”, //String
      “description”:”Ingresso digitale 3, overflow conteggio parziale”,
      “deviceDescription”:”IoT server-11x”,
      “dataType”:”BOOLEAN”, //String
      “devId”:3, //int
      “varId”:171, //int
      “value”:false, //Object(String,Boolean,Numeric)
      “quality”:true  //boolean
    }, //..... N variabili
  ],
  “devSn”: “IOTSPIXXXXXXXXX”,  //String
  “onTime”: “ Jan 8, 2018 10:50:34 AM”  //java.util.Date
}
```

2.2 Normal message

```
{
  “telemetryDataList”:[
    {
      “date”:”Jan 8, 2018 11:13:27 AM”,
      “devId”:3,
      “varId”:171,
      “value”:false,
      “quality”:true
    }, //..... N variabili
  ],
  “devSn”: “IOTSPIXXXXXXXXX”,
  “onTime”: “ Jan 8, 2018 11:13:27 AM “
}
```

2 Telemetry messages

The list of telemetryData, restored from IoT Scada, is a JSON consisting of:
- devId – identifier of the device, the variable belongs to (can be null in the case of plant variables)
- varId – identifier of the variable
- value – contains the value, read for the variable
- quality – boolean value, indicating whether last read attempt was successful or not, so if the value is up-to- date or potentially legacy
- date – date, last correct reading of the variable dates back to

Each message will have two header fields:
- devSn – contains the unique serial number of the device that forwards the variables
- onTime – forwarding date of the message

N.B. It is possible to distinguish these two types of messages (complete message and normal message) simply by ticking or not ticking in the IoT Scada interface “The forwarded message will contain only the essential information” entry.

2.3 Json Time series (JTS)

```
{
  “samples”:5,
  “samplesDelay”:0,
  “startTime”:1515406966223,
  “endTime”:1515406970224,
  “telemetryDataList”:[
    {
      “devId”:3,
      “varId”:171,
      “values”:[false, false, true, true, false],
      “quality”:[0,1,1,1,1]
    }, //..... N variabili
  ],
  “devSn”: “IOTSPIXXXXXXXXX”,
  “onTime”: “ Jan 8, 2018 11:13:27 AM “
}
```

The message contains the following header parameters:
- samples (s) – number of samples taken in the unit of time (can be set in MQTT Service interface);
- samplesDelay – Time interval of the single sampling (deriving from the formula (sT-eT)/s);
- startTime (sT) – start time of the sampling series;
- endTime (eT) – final timestamp of the sampling series.

N.B. It is possible to choose the following type of message, simply ticking “JTS (Json Time Series)”

entry, in IoT Scada interface. After that, it is possible to set the number of samplings, carried out in the time interval (Samples).

N.B. It is possible to choose the following type of message, simply ticking “JTS (Json Time Series)” entry, in IoT Scada interface. After that, it is possible to set the number of samplings, carried out in the time interval (Samples).

3 Questions to IoT Scada

As mentioned above, the IoT Scada can listen while waiting for messages, published on the broker, and respond promptly to requests for information.

The IoT Scada device will listen to the broker devSn/commands and respond to the requests devSn/telemetry.

All messages, recognized by the IoT Scada, have the same basic structure, to be completed in a different way, depending on the type of request.

The message recognized by IoT Scada has the following structure:

```
{
   “component”:””,
   “operation”:””,
   “devId”: [],
   “varId”: [],
   “startTime”:””,
   “endTime”:””,
   “value”:””
}
```

Evidently, as it will be explained later, some fields must be filled in for certain requests, and left empty for others, in case of non-recognition such message and response.

The basic structure consists of:

- component – (compulsory) - list of the involved components (INFO, DEVICES, ALARMS, EVENTS);
- operation – (compulsory, except INFO component) - list of the possible operations (LIST, DATA, CONFIG, LOGDATA, SET, ACTIVE, HISTORY);
- devId – (optional) – list of deviceIds;
- varId – (optional) – list of the variables;
- startTime - (optional) – initial endpoint of the time interval to extract data;
- endTime - (optional) – final endpoint of the time interval to extract data;
- value - (optional) – value to write in the variable.

3.1 System Information

3.1.1 Information about IoT Scada

To get such information, the JSON message to forward is:

```
{
   “component”:”INFO”
}
```

3 Questions to IoT Scada

The IoT Scada will respond with information related to it:

```
{
  “uuid”: “49ab91753-9b94-3d2e-8ff6-17d3d5113606”,
  “hwModel”: “4.0.5-SNAPSHOT”,
  “name”: “IoT Gateway”,
  “webAppVersion”: “4.0.3”,
  “devSn”: “ IOTSPIXXXXXXXXX “,
  “onTime”: “Sep 20, 2017 5:23:12 PM”
}
```

3.2 Configuration of connected devices

To get such information, forward the following JSON message

```
{
   “component”:”DEVICES”,
   “operation”:”LIST”,
   “devId”: [ 2, 3, 4 ]    //OPZIONALE
}
```

The parameter list devId is optional, it filters the information: if this parameter is not included, the device will provide the information from all devices. If it is set, only the information from the selected ones will be returned.

The IoT Scada will respond with a series of information regarding the device:

```
{
  “devices”: [
    {
      “devId”: 2,
      “description”: “Fanuc 750”,
      “linked”: true
    },
    {
      “devId”: 3,
      “description”: “Fanuc 545”,
      “linked”: true
    },
    {
      “devId”: 4,
      “description”: “MCM-cnc”,
      “linked”: false
    }
  ],
  “devSn”: “ IOTSPIXXXXXXXXX “,
  “onTime”: “Sep 20, 2017 5:23:12 PM”
}
```

4 Variables Information

4.1 Configuration of the variables

To get such information, forward the following JSON message:

```
{
   “component”:”DEVICES”,
   “operation”:”CONFIG”,
   “devId”: [ 63 ],    //OPZIONALE
   “varId”: [ 40, 41, 39 ]    //OPZIONALE
}
```

The parameter list devId is optional, it filters the information: if this parameter is not included, the device will provide the information from all devices. If it is set, only the information from the selected ones will be returned.

The parameter list varId is optional. If specified, there should be only one devId indicated in the corresponding parameter. The device will provide only the information regarding the selected varId for this device.

The IoT Scada will respond with a series of information regarding the device:

```
{
  “varConfigList”: [
    {
      “devId”: 63,
      “varId”: 21,
      “description”: “Actual executed program name”,
      “dataType”: “String”,
      “minimum”: “null”,
      “maximum”: “null”,
      “category”: [  “main”    ],
      “alarmable”: false,
      “writable”: false
    },
    {
      “devId”: 63,
      “varId”: 23,
      “description”: “Automatic mode selection”,
      “dataType”: “Numeric”,
      “minimum”: “null”,
      “maximum”: “null”,
      “category”: [  “main”    ],
      “alarmable”: false,
      “writable”: false
  ],
  “devSn”: “ IOTSPIXXXXXXXXX”,
  “onTime”: “Feb 20, 2017 4:11:13 PM”
}
```

4 Variables Information

4.2 Current data of the variables

To get such information, forward the following JSON message:

```
{
   “component”:”DEVICES”,
   “operation”:”DATA”,
   “devId”: [ 63 ],    //OPZIONALE
   “varId”: [ 23, 430 ]    //OPZIONALE
}
```

The parameter list devId is optional, it filters the information: if this parameter is not included, the device will provide the information from all devices. If it is set, only the information from the selected ones will be returned.

The parameter list varId is optional. If specified, there should be only one devId indicated in the corresponding parameter. The device will provide only the information regarding the selected varId for this device.

The IoT Scada will respond with a series of information regarding the device:

```
{
  “variablesList”: [
     {
       “devId”: 63,
       “varId”: 23,
       “value”: “MDI”,
       “decodedValue”: “MDI”,
       “date”: “Sep 20, 2017 5:48:57 PM”,
       “quality”: false
    },
    {
       “devId”: 63,
       “varId”: 430,
       “value”: false,
       “date”: “Sep 20, 2017 5:48:57 PM”,
       “quality”: false
    }
  ],
  “devSn”: “ IOTSPIXXXXXXXXX”,
  “onTime”: “Feb 20, 2017 4:29:46 PM”
}
```

4 Variables Information

4.3 Log Data

To get such information, forward the following JSON message:

```
{
   “component”:”DEVICES”,
   “operation”:”LOGDATA”,
   “devId”: [ 63 ],     //OBBLIGATORIO
   “varId”: [ 23 ],     //OBBLIGATORIO
   “startTime”: 0,    //OPZIONALE
   “endTime”: 123456789    //OPZIONALE
}
```

The parameters list devId are compulsory. In every list there must be only one value. The parameters startTime and endTime are optional. These parameters apply a temporary filter on the data to be restored. The values to be entered are expressed in milliseconds starting from 1/1/1070 0:00 GMT.

The IoT Scada will respond with a series of information regarding the variables:

```
{
        “devId”: 63,
        “varId”: 23,
        “value”: “MDI”,
        “date”: “Sep 8, 2017 1:37:14 PM”,
        “quality”: true
     },
     {
        “devId”: 63,
        “varId”: 23,
        “value”: “MDI”,
        “date”: “Sep 8, 2017 1:40:14 PM”,
        “quality”: true
     }
  ],
  “devSn”: “ IOTSPIXXXXXXXXX”,
  “onTime”: “Feb 20, 2017 5:02:49 PM”
}
```

devId and **varId** fields are optional. If left blank, the IoT Scada will respond with the list of all the variables; if one or more devIds are specified, the output will be filtered, returning only the requested ones.

If also a list of varIds is specified, the devId must be unique: only the variables with such devId and varId will be returned.

4 Variables Information

4.4 Setting a variable

To get such information, forward the following JSON message:

```
{
   “component”:”DEVICES”,
   “operation”:”SET”,
   “devId”: [ 31 ],    //OBBLIGATORIO
   “varId”: [ 47],      //OBBLIGATORIO
   “value”: 10     //OBBLIGATORIO
}
```

The parameters list **devId** are compulsory. In every list there must be only one value.
Value parameter indicates the value of the variable to be set.

The IoT Scada will respond with a series of information regarding the variables:

```
{
  “accepted”: true,
  “description”: “Accepted”,
  “devSn”: “ IOTSPIXXXXXXXXX”,
  “onTime”: “Feb 20, 2017 5:02:49 PM”
}
```

5 PartProgram Exchange

5.1 Upload file

In order to send a partprogram via MQTT to Alleantia, it is necessary to forward a JSON message like the one shown in the example:

```
{
   “component”:”DEVICES”,
   “operation”:”UPLOAD”,
   “devId”: [ 63 ],
   “partProgram”: {
   “requestId”: 1,
   “path”: “ROOT/”,
    “filename”: “prova.txt”,
    “content”:[10,-1,0,9]
}


}
```

The **requestID** field is an arbitrary request identifier useful for finding the associated response in the telemetry topic, the content is the byte array of the file you want to send
The IoT SCADA will respond with the following message:

```
{“accepted”:true,”description”:”requestid 1: prova.txt is upload correctly”,”devSn”:”44eb0bdc5-2b5d-3bb8-9d58-d7a9ccc6ef35”,”onTime”:”Jun 21, 2022 11:30:30 AM”,”ontTimeMillisUTC”:1655803830188}
```

5 PartProgram Exchange

5.2 Download file

In order to download a partprogram via MQTT from Alleantia, it is necessary to forward a JSON message like the one shown in the example:

```
{
   “component”:”DEVICES”,
   “operation”:”DOWNLOAD”,
   “devId”: [ 63 ],
   “partProgram”: {
   “requestId”: 10,
   “path”: “ROOT/prova.txt”
}


}
```

IoT SCADA will respond with the following message:

```
{“devices”:[{“devId”:194,”filename”:”prova.txt”,”content”:[10,-1,0,9]}
```

5.3 ListFile

In order to get the partprogram list via MQTT from Alleantia, it is necessary to forward a JSON message like the one shown in the example:

```
{
   “component”:”DEVICES”,
   “operation”:”LISTFILES”,
   “devId”: [ 194 ],
   “partProgram”: {
   “requestId”: 1,
   “path”: “ROOT/”
}


}
```

IoT SCADA will respond with the following message:

{“devices”:[{“devId”:194,”isFolder”:true,”path”:”ROOT/
sub/”,”length”:0},{“devId”:194,”isFolder”:false,”path”:”ROOT/prova.txt”,”length”:4}],”de
vSn”:”44eb0bdc5-2b5d-3bb8-9d58-d7a9ccc6ef35”,”onTime”:”Jun 20, 2022 3:22:43
PM”,”ontTimeMillisUTC”:1655731363436}

5.4 Delete

In order to delete a partprogram via MQTT the JSON message type is as shown in example:

```
{
   “component”:”DEVICES”,
   “operation”:”DELETE”,
   “devId”: [ 194 ],
   “partProgram”: {
   “requestId”: 1,
   “path”: “ROOT/prova.txt”
}


}
```

IoT SCADA will respond with the following message:

{“accepted”:true,”description”:”requestId 1: ROOT/prova.txt is deleted
correctly”,”devSn”:”44eb0bdc5-2b5d-3bb8-9d58-d7a9ccc6ef35”,”onTime”:”Jun 23, 2022 10:25:00
AM”,”ontTimeMillisUTC”:1655972700623}

5 PartProgram Exchange

5.5 Create SubFolder

In order to create a new folder for partprograms via MQTT the message JSON type is shownin example:

```
{
   “component”:”DEVICES”,
   “operation”:”CREATESUBFOLDER”,
   “devId”: [ 194 ],
   “partProgram”: {
   “requestId”: 1,
   “path”: “ROOT/”,
   “dirname”:”sub”
}


}
```

The IoT SCADA will respond with the following message:

```
{“accepted”:true,”description”:”requestId 1: the folder ROOT/sub is created
correctly”,”devSn”:”44eb0bdc5-2b5d-3bb8-9d58-d7a9ccc6ef35”,”onTime”:”Jun 23, 2022 10:26:10
AM”,”ontTimeMillisUTC”:1655972770346}
```

5 PartProgram Exchange

5.6 Delete SubFolder

In order to delete a subfolder the JSON message type to forward is this:

```
{
   “component”:”DEVICES”,
   “operation”:”DELETESUBFOLDER”,
   “devId”: [ 194 ],
   “partProgram”: {
   “requestId”: 1,
   “path”: “ROOT/sub”
}


}
```

The IoT SCADA will respond with the following message:

```
{“accepted”:true,”description”:”requestId 1: the folder ROOT/sub is deleted correctly”,”devSn”:”44eb0bdc5-2b5d-3bb8-9d58-d7a9ccc6ef35”,”onTime”:”Jun 23, 2022 10:29:09 AM”,”ontTimeMillisUTC”:1655972949642}
```

6 Alarm information

6.1 Alarm configurations

To obtain this information, the JSON message to be forwarded is:

```
{
   “component”:”ALARMS”,
   “operation”:”CONFIG”,
   “varId”: [ 47, 48]    //OPZIONALE
}
```

The varId list parameter is optional and is used to filter the information: omitting this parameter will cause the device to provide information from all alarms. By setting it, only the information of the selected ones will be returned.

The IoT Scada will respond with a set of event-related information:

```
{
  “alarmConfigList”: [
    {
      “id”: 48,
      “description”: “ALLARME CONTROTESTA”,
      “condition”: “$G_63_86 eq true”
    },
    {
      “id”: 43,
      “description”: “ALL.UTENSILI MOTORIZZATI”,
      “condition”: “$G_63_81 eq true”
    },
    {
      “id”: 47,
      “description”: “ALLARME CARICATORE”,
      “condition”: “$G_63_85 eq true”
    }

],
  “devSn”: “ IOTSPIXXXXXXXXX”,
  “onTime”: “Feb 20, 2017 5:25:48 PM”
}
```

6 Alarm information

6.2 Alarm status

To obtain this information, the JSON message to be forwarded is:

```
{
   “component”:”ALARMS”,
   “operation”:”DATA”,
   “varId”: [ 47, 48]    //OPZIONALE
}
```

The varId list parameter is optional and is used to filter the information: omitting this parameter will cause the device to provide information from all alarms. By setting it, only the information of the selected ones will be returned.

The IoT Scada will respond with a set of information about the alarms:

```
{
  “alarmDataList”: [
    {
      “id”: 48,
      “quality”: true,
      “alarmed”: false
    },
    {
      “id”: 43,
      “quality”: true,
      “alarmed”: true
    },
    {
      “id”: 47,
      “quality”: false,
      “alarmed”: false
    }
],
  “devSn”: “ IOTSPIXXXXXXXXX”,
  “onTime”: “Feb 20, 2017 5:25:48 PM”
}
```

7 Event information

7.1 Event configurations

To obtain this information, the JSON message to be forwarded is:

```
{
   “component”:”EVENTS”,
   “operation”:”INFO”,
   “varId”: [ 1 ]    //OPZIONALE
}
```

The varId list parameter is optional and is used to filter the information: omitting this parameter will provide the device with information from all events. By setting it, only the information of the selected ones will be returned.

The IoT Scada will respond with a set of event-related information:

```
{
  “eventsInfoList”: [
    {
      “eventId”: 1,
      “eventName”: “Nuovo Evento”,
      “type”: “boolean”,
      “condition”: “$G_63_71”,
      “snapshotGlobalIds”: “G_63_21”,
      “comparisonOperator”: “eq”,
      “numericCompareValue”: 1
    }
  ],
  “devSn”: “ IOTSPIXXXXXXXXX”,
  “onTime”: “Feb 20, 2017 5:25:48 PM”
}
```

7 Event information

7.2 Historical data on events

To obtain this informationthe JSON message to be forwarded is:

```
{
   “component”:”EVENTS”,
   “operation”:”HISTORY”,
   “varId”: [ 1 ],       //OBBLIGATORIO
   “startTime”:0,     //OPZIONALE
   “endTime”:123456789     //OPZIONALE
}
```

The varId list parameter is mandatory. There should be only one value in the list. The startTime and endTime parameters are optional. These parameters apply a time filter on the data to be returned. The values to be entered are in milliseconds from 1/1/1070 0:00 GMT

The IOT SCADA will respond with a set of event-related information:

```
{
  “eventHistoryList”: [
    {
      “eventId”: 1,
      “eventName”: “Nuovo Evento”,
      “timestamp”: “Sep 11, 2017 11:00:58 AM”,
      “state”: true,
      “variablesSnapshot”: [
        {
          “devId”: 63,
          “varId”: 21,
          “value”: “//CNC_MEM/USER/PATH1/TECNO”,
          “quality”: true
        }
      ]
    },
    {
      “eventId”: 1,
      “eventName”: “Nuovo Evento”,
      “timestamp”: “Sep 11, 2017 11:11:28 AM”,
      “state”: true,
      “variablesSnapshot”: [
        {
          “devId”: 63,
          “varId”: 21,
         “value”: “//CNC_MEM/USER/PATH1/O9110”,
          “quality”: true
        }
      ]
```

7 Event information

```
    },
  ],
  “devSn”: “ IOTSPIXXXXXXXXX”,
  “onTime”: “Feb 20, 2017 5:25:48 PM”
}
```

8 Other automated messages

Once the MQTT service is enabled on the IoT Scada, it will be able to automatically forward messages about new alarms and/or events to the broker, in addition to telemetry messages of course, as they occur.
Obviously, as with variables, in order for the service to notify them, any custom alarms and/or events must also first be enabled for forwarding in the IoT Scada interface.
The syntaxes of automatically forwarded messages are as follows.

8.1 New alarm raised

In case of a new alarm, the following JSON message will be forwarded to the topic configured for receiving alarms:

```
{
  “activeAlarmsList”: [
    {
      “id”: 11,
      “eventId”: 16153,
      “deviceName”: “Fanuc 545”,
      “measure”: “RIPARO APERTO”,
      “description”: “RIPARO APERTO”,
      “onDate”: “Sep 22, 2017 10:08:09 AM”,
      “offDate”: “Sep 22, 2017 10:10:25 AM”
    }
  ],
  “devSn”: “ IOTSPIXXXXXXXXX”,
  “onTime”: “Feb 20, 2017 5:25:48 PM”
}
```

The message will be forwarded twice: at the time of alarm activation, and will therefore obviously be absent the offDate parameter, and at the time of its return, and will be filled in as exactly as above.

The message will also contain the deviceSection field if present.

8 Other automated messages

8.2 New event raised

In case of a new event, the following JSON message will be forwarded to the topic configured for receiving events:

```
{
  “newEventsList”: [
    {
      “eventId”: 1,
      “eventName”: “Nuovo Evento”,
      “type”: “boolean”,
      “condition”: “$G_63_71”,
      “snapshotGlobalIds”: “G_63_21,G_63_820”,
      “comparisonOperator”: “eq”,
      “numericCompareValue”: 1,
      “timestamp”: “Sep 21, 2017 2:58:49 PM”,
      “snapshotVarsDatas”: [
        {
          “globalId”: “G_63_21”,
          “snapshotValue”: “//CNC_MEM/USER/PATH1/TECNO”
        },
        {
          “globalId”: “G_63_820”,
          “snapshotValue”: false
        }
      ],
      “eventValue”: “true”
    }
  ],
  “devSn”: “ IOTSPIXXXXXXXXX”,
  “onTime”: “Feb 20, 2017 5:25:48 PM”
}
```

There are two types of events: boolean, which are triggered when a condition occurs, and onChange, which are triggered when the value of a variable changes.

For the boolean event type, two messages will be forwarded to the broker: one at the time the condition occurs, and another at the time the condition is no longer true.

Events of the onChange type, by their nature, will forward a message toward the broker whenever the variable they monitor changes in value.

Note


Alleantia
ALL <THINGS> JOINED

Alleantia s.r.l.
www.alleantia.com

Sede legale: Via Tosco Romagnola, 136 - 56025 Pontedera (PI)
Sede operativa: Via G.Malasoma 26, 56121 Pisa
Partita IVA/Cod. fiscale: IT 02011550502

Tel: (+39) 050 9911933

M-MQTT-0622-EN