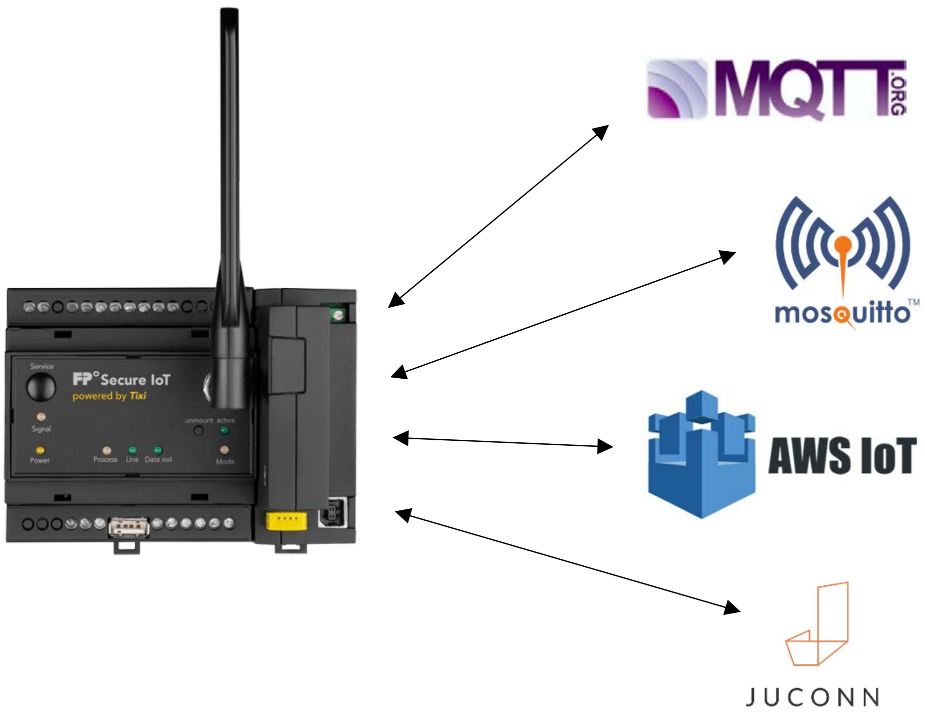
Universal MQTT Client for FP Gateways
User Manual



Version: 1.2.6

Publication date: 23/02/2021

Table of contents

1 Introduction

As of firmware version 5.2.6.14, a universal MQTT client can be configured in FP gateways.
The MQTT client can also be connected to any MQTT broker that supports the MQTT standard "MQTT 3.1.1".

1.1 Connection check

Up to 3 MQTT brokers can be configured in the client. The client then attempts to establish a connection to the first configured broker after the start. As soon as this broker fails, the client attempts to establish a connection to the second broker, etc. If the third broker also fails, the client starts again with the first broker.

1.2 Encrypted vs. unencrypted

The connection to the broker can either be established unencrypted (port 1883) or encrypted (port 8883). An encrypted connection can be established both to a public broker with public certificates and to a broker that works with private certificates. In the latter case, the required certificates can be stored in the gateway (see section 2).

1.3 Sending interval

As soon as an MQTT connection has been established, the client sends process data to the broker regularly. Up to 5 independent timers can be defined, which send different data to the broker cyclically or via a hysteresis function.
The sending interval can defined in a wide range (default: 3 seconds).
There is also a feedback channel that the broker can use to send TiXML commands to the client.
The results of the commands are then sent back to the broker via another topic.

1.4 MQTT topics

The topic path to send data to the broker (publish topic) comprises a fixed part (**`/asset/telemetry/`**) and a variable part (**`asset_id`**):
`/asset/telemetry/asset_id`

Example: In the configuration, the **`Property1`** asset-id is configured.
This results in the following publish topic: **`/asset/telemetry/Property1`**
to which the client sends the realtime data.

The data format is explained in the following sections.

Overview of topics used

Topic	Direction	Meaning
/asset/telemetry/asset_id	publish	Send data to the broker asset_id can be configured
/asset/commands/asset_id	subscribe	The broker can use this topic to send commands to the client
/asset/commandresp/asset_id	Publish	The client sends the command execution result back to the broker

Table 1-1: Overview of topics used

2 TiXML configuration for the MQTT client

The MQTT client is configured using a TiXML database.

```
[<SetConfig _="ISP" ver="y">

   <CloudConn>

     <!-- connection without tls/ssl -->
     <CloudBaseUrl _="tcp://URL_or_IP_of_MQTT_broker:1883"/>

     <!-- optional: up to two additional URLs definable -->
     <CloudBaseUrl2 _="tcp://2nd_URL_or_IP_of_MQTT_broker:1883"/>
     <CloudBaseUrl3 _="tcp://3rd_URL_or_IP_of_MQTT_broker:1883"/>

     <username _="user" />
     <password _="password" />

     <!-- asset ID, e.g. ID of the device; used in subscribe topics,
          e.g. "asset/telemetry/asset_id" -->
     <asset_id _="Property1" />

     <!-- optional: QoS (quality of service setting)
         valid values: 0 = send once (not guaranted); Default
                       1 = send at least once (guaranted)
                       2 = send only once (guaranted)
         Please note that not all brokers supports QoS 1 and 2
         (e.g. AWS:  supports QoS=1 only)    -->
     <QoS _="1" />

     <!-- optional: client authentication setting.
```

TLS_Client_authmode	Description
file	keys and certificates will be stored in internal flash memory. Path: /flash_user/app/VPN files to be used are defined with TLS_Client_key and TLS_Client_cert
Hsm	(Default) keys and certificates will be stored in hardware security module

```
     -->
     <TLS_Client_authmode _="file" />

     <!-- optional: key and cert files -->
     <!-- The following parameters will only be used if
          TLS_Client_authmode was set to "file" -->
     <TLS_Client_key _="mykey.key" />
     <TLS_Client_cert _="myclientcert.crt" />
     <!-- optional: server certificate.
```

TLS_Server_authmode	Description
cabundle none	no server authentication (Default)
cabundle	(Default) internal Mozilla root certs bundle file
file	server certificate will be stored in internal flash memory. Path: /flash_user/app/VPN TLS_Server_cert (see below)
hsm	server certificate will be stored in hardware security module

```
     -->
```

```
    <TLS_Server_authmode _="file" />

    <!-- optional: key and cert files -->
    <!-- The following parameters will only be used if
         TLS_Client_authmode was set to "file" -->
    <TLS_Server_cert _="servercert.crt" />


    <!-- MQTT Keep Alive interval in seconds (default = 300) -->
    <MQTTKeepAlive _="200"/>

    <!-- if set to 1 the MQTT connection will be started at device
         startup; otherwise MQTT connection will not be established -->
    <CloudConnStart _="1"/>

    <!-- fixed setting (mandatory) -->
    <CloudConnType _="TixiSimpleMQTT_1"/>

    <!-- RTSendTimeX and UpdateRate1 controls the send interval
         X = 1 .. 5
         RTSendTimeX ranges:
          RTSendTimeX = -1: do not send data using a time
                            or activate hysteresis (RTSendTime1 only)
          RTSendTimeX = 1 .. 2^30: send interval in seconds
          RTSendTimeX = empty: send interval = 3 seconds (default)

         UpdateRate1 ranges (only used when RTSendTime1 = -1)
          UpdateRate1 = 1 .. 3600 = update interval of data
                                    points in seconds

         Hysteresis is defined within the RealTime1 section (below).
         In hysteresis mode the variables will only be sent to the
         broker if a pre-defined treshhold is reached
         After a connection to the broker has been established all
         variables of a hysteris RealTime1-section will be sent once.

         hysteresis ranges: floating point number, max. two digits
         after the decimal point. Examples: 0.1 / 2 / 2.68
    -->

    <!-- specifies the first send timer, operating mode = hysteresis -->
    <RTSendTime1 _="-1" />

    <!-- update rate for variables in hysteresis mode in seconds -->
    <UpdateRate1 _="20" />

    <!-- list of variables for first send timer to be sent to broker
         The gateway reads out the current values from the Process
         tree using the UpdateRateX. If a data point has changed and
         the changes are bigger than the specified hysteresis the data
         point will be sent to the broker.
    -->
    <Realtime1>
      <Datapoint1 _="/Process/Bus1/Device1/Datapoint1" hysteresis="2"/>
      <Datapoint2 _="/Process/Bus1/Device1/Datapoint2" hysteresis="1"/>
      <Datapoint3 _="/Process/Bus1/Device2/Datapoint3" hysteresis="5"/>
      <Datapoint4 _="/Process/PV/ProcVar1" hysteresis="0.4"/>
      <Datapoint5 _="/Process/PV/ProcVar2" hysteresis="1.7"/>
    </Realtime1>
```

```
        <!-- specifies the second send timer, send interval = 10 seconds -->
        <RTSendTime2 _="10" />
        <Realtime2>
           <Obis_180 _="/Process/Meter/MT174/Obis_180"/>
        </Realtime2>

        <!-- specifies the third send timer, no send interval -->
        <!-- data can be sent event driven using CloudSendRealtimeData -->
        <RTSendTime3 _="-1" />
        <Realtime3>
           <Obis_181 _="/Process/Meter/MT174/Obis_181"/>
        </Realtime3>

        <!-- specifies the forth send timer, send interval = 60 seconds -->
        <RTSendTime4 _="60" />
        <Realtime4>
           <Obis_182 _="/Process/Meter/MT174/Obis_182"/>
        </Realtime4>

        <!-- specifies the fifth send timer, send interval = 100 seconds -->
        <RTSendTime5 _="100" />
        <Realtime5>
           <Temp_Board _="/Process/OneWire/Mainboard/TEMP_Board"/>
        </Realtime5>
        .
        .
        <!-- specifies the fifth send timer, send interval = 100 seconds -->
        <RTSendTime5 _="100" />
        <Realtime5>
           <Temp_Board _="/Process/OneWire/Mainboard/TEMP_Board"/>
        </Realtime5>

      </CloudConn>

</SetConfig>]
```

Please note the following information:

- The configuration options shown in bold must be configured (mandatory).
- At least one CloudBaseUrl must be configured.
- The path for unencrypted URLs is: `tcp://URL-or_IP_of_MQTT_broker:1883`
- The path for encrypted URLs is: `ssl://URL-or_IP_of_MQTT_broker:8883`
- Up to 3 CloudBaseUrls can be configured.
- The asset_id must be configured. Maximum text length: 50 characters.
  References are also supported.
  Example: Device serial number as asset_id:
  `<asset_id _="&#xae;/SerialNo" />`
- CloudConnStart and CloudConnType must be configured as described above.
- The data in the branches Realtime1 - RealTime5 can be configured freely by the user.

After implementing the MQTT client configuration, the FP gateway must be restarted.
You can monitor the MQTT client’s connection status in the process branch:
`[<Get _="/Process/CloudConn/" ver="y" />]`

Result (example):

```
<Get>
  <CloudConn>
    <ConnectionType _="mqtt+ssl:172.18.121.247:8883"/>
    <ConnectionState _="1"/>
    <ConnectionStateMsg _="connected"/>
    <LastTimeStamp _="Timestamp"/>
    <ChangeToggle _="Togglebit"/>
  </CloudConn>
</Get>
```

`ConnectionType` indicates the type of connection (unencrypted = mqtt/encrypted = mqtt+ssl), the URL / IP address for the broker that is currently active and the MQTT port (e.g. 1883 or 8883).
`ConnectionState` = 0 means: no connection
`ConnectionState` = 1 connection is established
`ConnectionStateMsg` = (registering, not connected, connecting, connected)
`LastTimeStamp` = The time stamp indicates when the last MQTT message was sent.
Format: DD.MM.YYYY HH:MM:SS time zone
`ChangeToggle` = (switches between 0 and 1 each time when the time stamp has changed)

2.1 Realtime sections

The data points that are to be transmitted can be defined in up to 5 realtime sections.

The cycle for each realtime section must be preset. The cycle time is also used to define whether the data is transmitted cyclically or via a hysteresis function.

E.g. `RTSendTime1, …, RTSendTime5.`
The data from the corresponding sections <RealTime1>..</RealTime1>, etc. then belongs to this.

Example:

```
<CloudConn>
    <RTSendTime1 _="-1" />
    <UpdateRate1 _=20 />
    <Realtime1>
       <Volt_L1_N _="/Process/Modbus/D1/Volt_L1_N" hysteresis="2" />
       .
       .
    </Realtime1>

    <RTSendTime2 _="10" />
    <Realtime2>
       <Obis_180 _="/Process/Meter/MT174/Obis_180"/>
    </Realtime2>
    .
    .
    <RTSendTime5 _="100" />
    <Realtime5>
       <Temp_Board _="/Process/OneWire/Mainboard/TEMP_Board"/>
    </Realtime5>
</CloudConn>
```

2.2 Sending data via the EventHandler or command

Data points can also be sent via EventHandler or by command:
Command (TICO, ...):

```
 <CloudSendRealtimeData [_="CloudConn[,Realtime_Specifier"]] />
```
or
```
 [<CloudSendRealtimeData />]
```

CloudConn (optional) = name of the CloudConnector. Values: CloudConn, CloudConn2, CloudConn3.
Default: CloudConn

Realtime_Specifier (optional) = name of the realtime section, e.g. Realtime1. Default=Realtime1
If the optional parameter is omitted, all defined sections are sent.

Example 1: Send Realtime1 sections from CloudConn
```
 <CloudSendRealtimeData />
```

Example 2: Send realtime section "Realtime2" from CloudConn2
```
 <CloudSendRealtimeData _="CloudConn2,Realtime2" />
```

Example 3: Send realtime section "Realtime3" from CloudConn
```
 <CloudSendRealtimeData _="CloudConn,Realtime3" />
```

Example 4: Send realtime section "Realtime2" from CloudConn2 via an EventHandler
```
 <CloudSendRealTimeDate_2_2>
       <CloudSendRealtimeData _="CloudConn2,Realtime2" />
      </CloudSendRealTimeDate_2_2>
```

Minute-by-minute execution of the EventHandler via scheduler:
```
 <CloudSendRealTimeDate_2_2_SCH _="CloudSendRealTimeDate_2_2">
       <Minute _="0-59" />
 </CloudSendRealTimeDate_2_2_SCH>
```

Test via TiXML command with TICO
```
 [<DoOn _="CloudSendRealTimeDate_2_2" ver="v" />]
```

2.3 Hysteresis

For Timer 1 (`RTsendTime1`), there is the option to set whether each data point is to be transmitted depending on a hysteresis setting.

Example: Data point “Temperature”

Start value	= 20.5K
Absolute hysteresis	= 0.5

Result: If the data point’s value changes to 20 or 21, this is transmitted.

Regulations:
- All values are transmitted once after restarting the device.
- If the last value that was transmitted changes by the hysteresis amount, the value is transmitted once again.

The required parameters:

```
<RTsendTime1 _="-1" />
```
The value `-1` means that data is no longer sent cyclically but using the hysteresis parameters that were set.

```
<UpdateRate1 _="Polltime" />
```
Poll time specifies the frequency with which the values are requested from the process branch. The poll time is specified in seconds. Value range: 1 .. 3600

Attention:
The hysteresis function is currently only available for the first timer(`RTsendTime1` and `UpdateRate1`). The hysteresis function can only be used for numeric values.

The hysteresis is defined as a floating comma value in the configuration for each realtime value.
Example:

```
hysteresis="1.5"
```

Definition:

- The comma / decimal point is displayed with "."
- Max. 2 decimal places
- No negative numbers

Example:

```
<RTsendTime1 _="-1" />
<UpdateRate1 _="20" />

<Realtime1>
   <Voltage_L1_N _="/Process/Modbus/D1/Voltage_L1_N" hysteresis="2" />
   <CPULoad _="/Process/MB/CPULoad" hysteresis="8.5" />
   <SupplyTemp _="/Process/PV/SupplyTemp" hysteresis="0.5" />
   <ReturnTemp _="/Process/PV/ReturnTemp" hysteresis="0.8" />
</Realtime1>
```

In the example above, the hysteresis is defined for the individual data points.
If the data point `Voltage_L1_N` changes by +/- 2.0 compared to the last value that was transmitted, this value is transmitted. If the data point `CPULoad` changes by +/- 8.5 compared to the last value that was transmitted, this value is transmitted, etc.

2.4 MQTT connection visual indication

The “Signal” LED indicates the MQTT connection status.
Off = no MQTT connection active
Flashing red = MQTT connection is being established
Illuminated green = MQTT connection is established

2.5 Service routing

The connection to the broker is normally established using the active LAN interface.
The TiXML database `ISP/ISP/OUT` can also be used to establish the MQTT connection via a mobile communications connection or a VPN tunnel:

```
[<SetConfig _="ISP/ISP" ver="y">
    <!-- Define communication interface for services -->
    <OUT>
        <SMTP _="MODEM"/>
        <CBIS _="MODEM"/>
        <POP3 _="MODEM"/>
        <URLSend _="MODEM"/>
        <INetTime _="MODEM"/>
        <HTTPConn _="MODEM"/>
        <CloudConn _="Ethernet"/>
        <IBMConn _="MODEM"/>
        <FTPPut _="MODEM"/>
        <SFTPPut _="MODEM"/>
        <VPN _="MODEM"/>
    </OUT>
</SetConfig>]
```

The following options can be used for the MQTT client:

```
Ethernet
MODEM
VPN
```

3 Data format for the process data

The MQTT client sends the process data as an MQTT payload in the following format:

```
<Data>
    <TimeStamp _="03.04.2018 11:48:00 +0100" />
    <Datapoint1 _="1" flag="1"/>
    <Datapoint2 _="20" flag="1"/>
    <Datapoint3 _="75.7" flag="1"/>
    <Datapoint4 _="13" flag="2"/>
    <Datapoint5 _="189" flag="2"/>
</Data>
```

The time stamp is formed from the current system time and the time zone configured in the USER database (`/USER/USER/TimeZone`).

The `flag` provides information regarding the variable type.
`flag` = 0 or 1: indicates the DeviceState for a **bus variable**. (0 = data invalid, 1 = data valid)
`flag` = 2: this is not a **bus variable** but, for example, a process variable or a variable from an expansion module (e.g. “`/Process/C621/Q0`”).

According to the configuration of the timer `RTSendTime1` - `RTSendTime5`, the process data is sent to the broker.

4 Feedback channel to the FP gateway

The MQTT broker can use a feedback channel to send TiXML commands to the FP gateway in order to save a new configuration to the device, to restart the device or to switch an output for example. The MQTT broker can send all TiXML commands that are also possible using the TICO configuration software to the device. The client responds to the commands using a special publish topic.

4.1 Sending a command

The MQTT client subscribes (subscribe) to the topic `/asset/commands/asset_id`

The broker can now send any TiXML commands to the MQTT client:

```
<Command>
    <!-- The RequestID must be generated by the broker
         It can be any string containing letters and numbers.
         Max. Length: 30 characters
         The RequestID will be sent back by the FP gateway
         together with the response to the TiXML command. -->

    <RequestID _="ID" />
    <!-- Put your command within the TiXML tag.
         It can be any TiXML command (without the command frame [])
         including Set, SetConfig, GetConfig, etc.
         The FP gateway will execute the command and sends
         back the response using the topic asset/commandresp/asset_id -->

     <TiXML>
     <!-- Here comes the TiXML command...
          Example: SetConfig of AccRights Database
     -->
        <SetConfig _="USER" ver="y">
           <AccRights>
              <Groups>
                 <Admin>
                   <LocalLogin AccLevel="1"/>
                   <CardLogin AccLevel="1"/>
                   <RemoteLogin AccLevel="1"/>
                   <EthernetLogin AccLevel="1"/>
                 </Admin>

                 <Webserver>
                   <WebServer AccLevel="20"/>
                 </Webserver>
              </Groups>

              <User _="Plain">
                   <Def_LocalLogin Plain="" Group="Admin"/>
                   <Def_CardLogin Plain="" Group="Admin"/>
                   <Def_RemoteLogin Plain="" Group="Admin"/>
                   <Def_EthernetLogin Plain="" Group="Admin"/>
                   <ADMIN Plain="" Group="Admin"/>
              </User>
           </AccRights>
        </SetConfig>

     </TiXML>
</Command>
```

4.2 Receiving the result of a command

The client responds to the commands using the publish topic `/asset/commandresp/asset_id`
To do this, the client sends the response to the broker in the following format:

```
<CommandResp>
    <!-- The FP gateway sends back the command response together with
         the RequestID from the command sent by the broker.
         It can be any string containing letters and numbers.
         Max. Length: 30 characters
         The RequestID will be sent back by the FP gateway
         together with the response to the TiXML command. -->

    <RequestID _="ID" />

    <!-- Here comes the command response from the FP gateway.
         The response is exactly what you see in TICO, except the
         command frame ([ ] will be omitted).
    -->

     <TiXML>
      <!-- Here comes the TiXML command response ...
           Example: SetConfig of AccRights Database was sent
           Response is an empty SetConfig command or an error description
           in case the command execution failed.
      -->
         <SetConfig/>

      </TiXML>
</CommandResp>
```

4.3 Gateway error messages

If the gateway receives commands from the broker and executes these commands, error messages are sometimes generated if the command contains incorrect parameters for example, or cannot be executed for other reasons.

A parameter can be used when sending the command to specify how comprehensive the response to the gateway error messages is.

There are three possible formats that can be used to transmit these errors. A short or a long error message is issued depending on the specification by the “ver” parameter. See Table 4-1.

Command example:

```
    [<Get _="/Process/" ver="vmode"/>]
```

vmode	Description	Example
n	Error message as numeric value	`<Error _="-2196" />`

y	Error message, short description	`<Error>`
		`  <ErrNo _="-2196" />`
		`  <ErrText _="path to key not found" />`
		`  <ErrorCause>`
		`    <ErrNo _="-2196" />`
		`    <ErrText _="path to key not found" />`
		`    <Class _="TXSTCPGetSetValueCmd" />`
		`  </ErrorCause>`
		`</Error>`

v	Error message, long description	`<Error>`
		`  <ErrNo _="-2196" />`
		`  <ErrText _="path to key not found" />`
		`  <ErrorCause>`
		`    <ErrNo _="-2196" />`
		`    <ErrText _="path to key not found" />`
		`    <Line _="127" />`
		`    <Module _="SSet.cpp" />`
		`    <Class _="TXSTCPGetSetValueCmd" />`
		`  </ErrorCause>`
		`</Error>`

Table 4-1: Error return values depending on the “ver” parameter

Important:
Before each configuration command (i.e. all `SetConfig` commands), internal processing should always be stopped using the following command:

```
<Command>
    <RequestID _="ID" />
    <TiXML>
      <Set _="/Process/Program/Mode" value="Stop" ver="v"/>
    </TiXML>
</Command>
```

Wait for the result of the aforementioned command. A `SetConfig` command should only be executed once the aforementioned stop command has been executed successfully.
If no response to the stop command is received within 15 seconds, the stop command must be sent again.
After the configuration database(s) was/were sent to the device, the following command can be used to restart process editing:

```
<Command>
    <RequestID _="ID" />
    <TiXML>
      <Set _="/Process/Program/Mode" value="Run" ver="v"/>
    </TiXML>
</Command>
```