SIEMENS

Ingenuity for life

# MQTT Publisher (unencrypted) for the S7-1500/ S7-1200 and S7-300

Blocks for S7-1500/ S7-1200, S7-300, Version 1.1

https://support.industry.siemens.com/cs/ww/en/view/109748872

Siemens Industry Online Support

# Legal information

# Table of contents

# 1 Introduction

## 1.1 Overview

### Motivation

Digitization has a major impact on the economy and society and is progressing inexorably. The "Internet of Things", short: IoT) is one of the main drivers of digitization. The term "Internet of Things" is synonymous with one of the biggest current dynamics of change: The increasing networking and automation of devices, machines and products.

The protocol "Message Queue Telemetry Transport" (short: MQTT) is used in the "Internet of Things" as a communication protocol. Its lightweight approach opens up new possibilities for automation.

### Slim and quick MQTT

The MQTT is a simple built-in binary publish and subscribe protocol at the TCP/IP level. It is suitable for messaging between low-functionality devices and transmission over unreliable, low-bandwidth, high-latency networks. With these characteristics, MQTT plays an important role for IoT and in M2M communication.

### Features of MQTT

The MQTT protocol is distinguished by the following features:

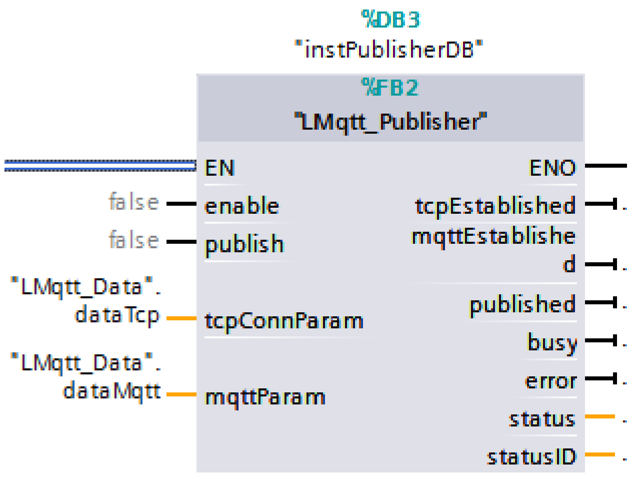
- Lightweight protocol with low transport overhead
- Minimal need for network bandwidth through push mechanism
- Reconnect function after termination of connection
- Resending of messages after disconnection
- Mechanism to notify interested parties of an unexpected connection abort of a client
- Easy to use and implement with a small set of command commands
- Quality Assurance (QoS level) with different levels of message delivery reliability
- Optional encryption of messages with SSL/TLS
- Authentication of publishers and subscribers with username and password

### Applicative implementation

This application example offers you an adequate solution for implementing the MQTT protocol in a SIMATIC S7 controller.

The application example provides you with one function block each for the SIMATIC S7-1500/ SIMATIC S7-1200 and for the SIMATIC S7-300. The function module "LMqtt_Publisher" integrates the MQTT client function and allows you to transfer MQTT messages to a broker (publisher role).

Figure 1-1



**Note** The MQTT client supports MQTT protocol version 3.1.

## 1.2 How it works

### Schematic representation

The following figure shows the most important relationships between the components involved and the steps required for MQTT communication.

Figure 1-2

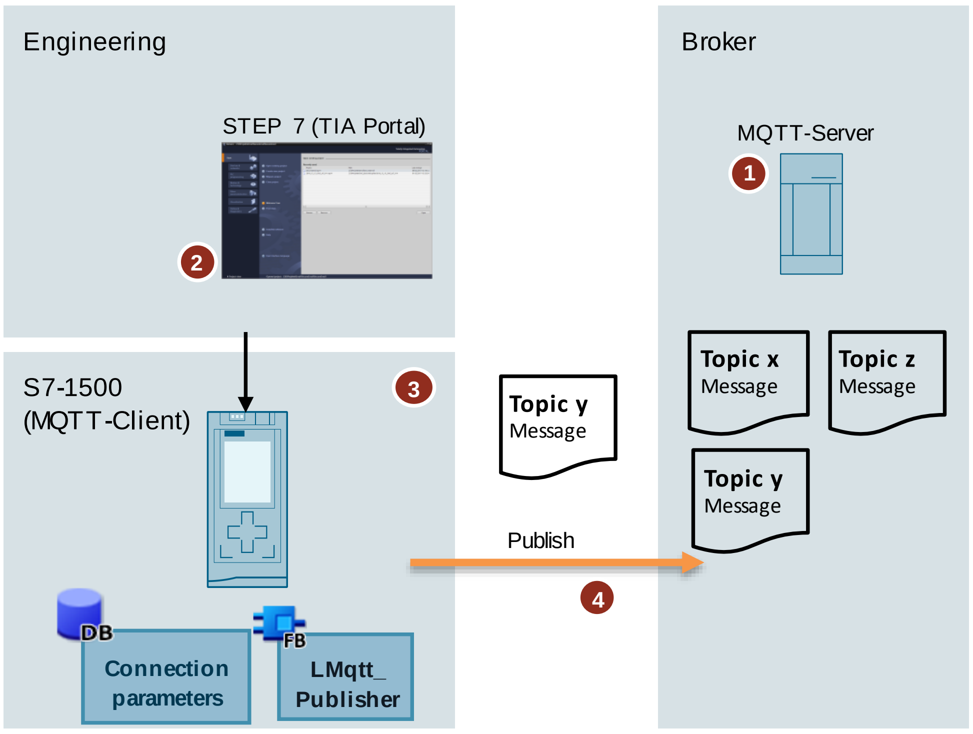


Table 1-1

| Step | Description |
|---|---|
| 1 | Install and configure the MQTT broker. |
| 2 | Create a project in STEP 7 (TIA Portal) with your CPU. |
| 3 | The function module "LMqtt_Publisher" takes over the role of the publisher and sends MQTT messages to the broker. |
| 4 | The MQTT message is stored in the broker and distributed to the subscribers. |

**Note** A more detailed functional description of the function block "LMqtt_Publisher" and information on the MQTT protocol can be found in Chapter 3.

## 1.3 Components used

The application example was created with these hardware and software components:

Table 1-2

| Component | Number | Article number | Note |
|---|---|---|---|
| CPU 1513-1 PN | 1 | 6ES7513-1AL01-0AB0 | You can also use a different CPU. |
| CPU 317-2 PN/DP | 1 | 6ES7317-2EK14-0AB0 | You can also use a different CPU. |
| TIA Portal V15 | - | - | |
| MQTT broker | - | - | |

This application example consists of the following components:

Table 1-3

| Component | File name |
|---|---|
| Library "LMqtt" and "LMqttQdn" for SIMATIC S7-1500 | 109748872_MqttClient_Publish_Unsecure_S71500_LIB_V1_1.zip |
| Library "LMqtt" for SIMATIC S7-300 | 109748872_MqttClient_Publish_Unsecure_S7300_LIB_V1_1.zip |
| This document | 109748872_MqttClient_Publish_Unsecure_DOKU_V1_1_en.pdf |

**Note** To reach the broker over a static ip address, please use the library "LMqtt".
To reach the broker over a Qualified Domain Name (short: QDN), please use the library "LMqttQdn".

# 2 Engineering

**Note** The engineering in this chapter focuses on the MQTT client function, which realizes this application example.
It is assumed that you have already installed and configured the MQTT broker.

## 2.1 Block description

### 2.1.1 Interface description "LMqtt_Publisher"

**Note** The function block "LMqtt_Publisher" is available in all libraries and is equal.

For the S7-1500, the function block is designed for "optimized block access".

The following section explains the input and output parameters of the function block "LMqtt_Publisher".

#### Input parameters

Table 2-1

| Parameter | Data type | Function |
|---|---|---|
| enable | BOOL | The function block is activated with a positive edge. The function block is active as long as "enable" has the status "true".<br>A negative edge terminates the function block and the TCP and MQTT connection is terminated. |
| publish | BOOL | A message is sent to the broker with a positive edge. |
| tcpConnParam | "typeTcpConnParam" | Data area of the TCP connection information |
| mqttParam | "typeMqttParam" | Data area of the MQTT connection and message information |

#### Output parameters

Table 2-2

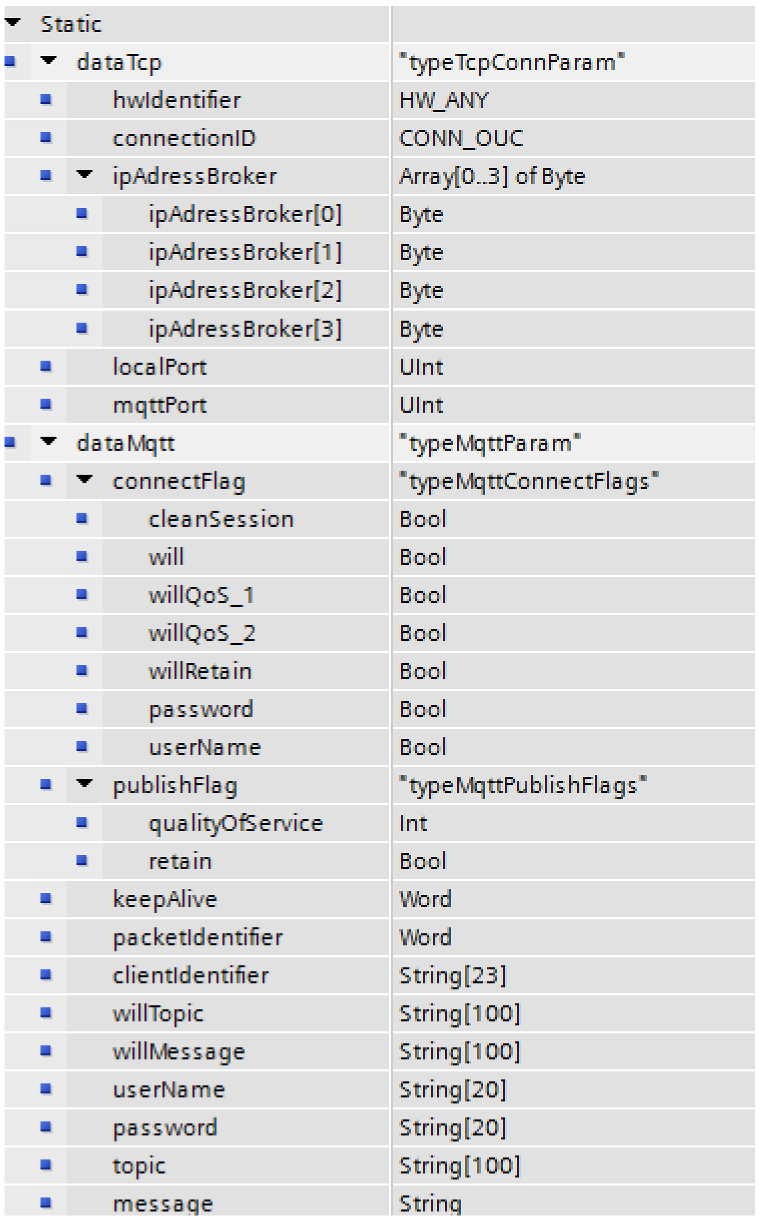
| Parameter | Data type | Function |
|---|---|---|
| tcpConnected | BOOL | True if the TCP connection has been established |
| mqttConnected | BOOL | True if the MQTT connection has been established |
| published | BOOL | True, if the message has arrived successfully at the broker. Only one cycle is on "true". |
| busy | BOOL | True, while a message or ping is sent to the broker |
| error | BOOL | True if there is an error |
| statusID | INT | State that caused the error |
| status | DWORD | Error message |

## 2.1.2 Data block "LMqtt_Data"

The following figure illustrates the declaration of the data block for the SIMATIC S7-1500/ SIMATIC S7-1200:

**Note** The data block is designed for "optimized block access".

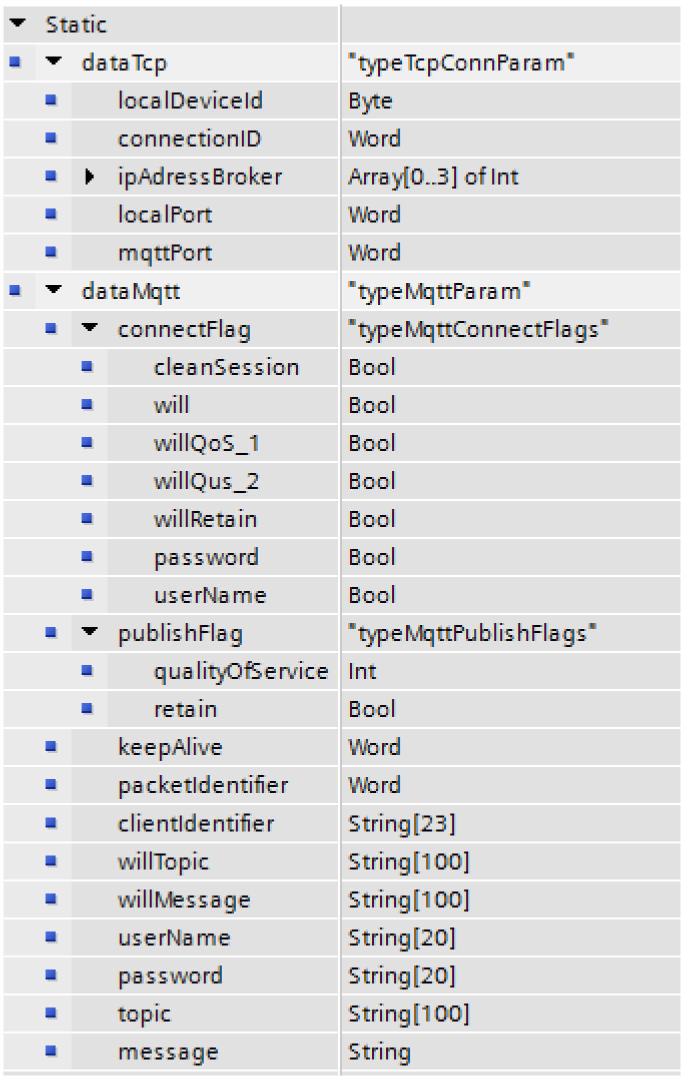
Figure 2-1



| | |
|---|---|
| Static | |
| dataTcp | "typeTcpConnParam" |
| hwIdentifier | HW_ANY |
| connectionID | CONN_OUC |
| ipAdressBroker | Array[0..3] of Byte |
| ipAdressBroker[0] | Byte |
| ipAdressBroker[1] | Byte |
| ipAdressBroker[2] | Byte |
| ipAdressBroker[3] | Byte |
| localPort | UInt |
| mqttPort | UInt |
| dataMqtt | "typeMqttParam" |
| connectFlag | "typeMqttConnectFlags" |
| cleanSession | Bool |
| will | Bool |
| willQoS_1 | Bool |
| willQoS_2 | Bool |
| willRetain | Bool |
| password | Bool |
| userName | Bool |
| publishFlag | "typeMqttPublishFlags" |
| qualityOfService | Int |
| retain | Bool |
| keepAlive | Word |
| packetIdentifier | Word |
| clientIdentifier | String[23] |
| willTopic | String[100] |
| willMessage | String[100] |
| userName | String[20] |
| password | String[20] |
| topic | String[100] |
| message | String |

The following figure illustrates the declaration of the data block for the SIMATIC S7-300:

Figure 2-2



| | |
|---|---|
| Static | |
| dataTcp | "typeTcpConnParam" |
| localDeviceId | Byte |
| connectionID | Word |
| ipAdressBroker | Array[0..3] of Int |
| localPort | Word |
| mqttPort | Word |
| dataMqtt | "typeMqttParam" |
| connectFlag | "typeMqttConnectFlags" |
| cleanSession | Bool |
| will | Bool |
| willQoS_1 | Bool |
| willQus_2 | Bool |
| willRetain | Bool |
| password | Bool |
| userName | Bool |
| publishFlag | "typeMqttPublishFlags" |
| qualityOfService | Int |
| retain | Bool |
| keepAlive | Word |
| packetIdentifier | Word |
| clientIdentifier | String[23] |
| willTopic | String[100] |
| willMessage | String[100] |
| userName | String[20] |
| password | String[20] |
| topic | String[100] |
| message | String |

## Overview of data types

To structure the data clearly, several data types have been created. The data types used in the program are shown in the following list:

- "typeTcpConnParam"
- “typeMqttParam"; divided into
  - "typeMqttConnectFlags"
  - "typeMqttPublishFlags"

## Data type "typeTcpConnParam"

This data type stores all information required to establish the TCP connection. You can set these parameters according to your specifications.

The following table displays the parameters of the SIMATIC S7-1500/ SIMATIC S7-1200:

Table 2-3

| Parameter | Data type | Meaning |
|---|---|---|
| hwIdentifier | HW_ANY | HW ID of the PROFINET interface of the CPU |
| connectionID | CONN_OUC | ID of the TCP connection |
| ipAdressBroker | Array[0..3] of BYTE | IP address of the broker, e.g. for the address 192.168.0.10.<br>ipAdressBroker[0] equal to "192"<br>ipAdressBroker[1] equal to "168"<br>ipAdressBroker[2] equal to "0"<br>ipAdressBroker[3] equal to "10" |
| localPort | UINT | Local port number in the CPU |
| mqttPort | UINT | Remote port on the MQTT broker |

**Note** If you use the blocks from the library "LMqttQdn", then you find the parameter "qdnAddressBroker" instead of the parameter "ipAddressBroker".

The following table displays the parameters of the SIMATIC S7-300:

Table 2-4

| Parameter | Data type | Meaning |
|---|---|---|
| localDeviceID | Byte | Slot designation of the PROFINET interface of the CPU (see Chapter 4.2) |
| connectionID | CONN_OUC | ID of the TCP connection |
| ipAdressBroker | Array[0..3] of INT | IP address of the broker, e.g. for the address 192.168.0.10.<br>ipAdressBroker[0] equal to "192"<br>ipAdressBroker[1] equal to "168"<br>ipAdressBroker[2] equal to "0"<br>ipAdressBroker[3] equal to "10" |
| localPort | UINT | Local port number in the CPU |
| mqttPort | UINT | Remote port on the MQTT broker |

**Data type "typeMqttParam"**

This data type contains all the information about MQTT. The information you can store here is shown in the following list:

- Flags for the connection
- Flags for sending messages
- Logon information at the broker
- Topic
- Message text

To display the large number of parameters in a more structured way, separate data types have been created for the flags.

With the data type "typeMqttConnectFlags" you can determine the flags for establishing the connection to the MQTT broker.

Table 2-5

| Parameter | Data type | Meaning |
|---|---|---|
| cleanSession | BOOL | True if all data from a previous session should be deleted. |
| will | BOOL | Activates the "Last Will and Testament" feature. |
| willQoS_1 | BOOL | True if the QoS for last will is Level 1. |
| willQoS_2 | BOOL | True if the QoS for last will is Level 2. |
| willRetain | BOOL | True if the last will be saved as soon as it's sent. |
| password | BOOL | True if the MQTT broker requires a login (name and password) of the client. |
| username | BOOL | True if the MQTT broker requires a login (name and password) of the client. |

You can use the data type "typeMqttPublishFlags" to determine the flags for the MQTT message.

Table 2-6

| Parameter | Data type | Meaning |
|---|---|---|
| qualityOfService | INT | Defines the QoS level for the MQTT message. Possible values are:<br>• "0" for QoS level 0<br>• "1" for QoS level 1<br>• "2" for QoS level 2 |
| retain | BOOL | True if the message should be saved to the broker. |

The following table shows which other parameters of the data type "typeMqttParam" you can specify for MQTT.

Table 2-7

| Parameter | Data type | Meaning |
|---|---|---|
| keepAlive | WORD | Time interval of the KeepAlive function in seconds. The time is given in hexadecimal format. A keepAlive with value "0" disables the KeepAlive function. The maximum allowed time is 18h 12min 15 s. |
| packetIdentifier | WORD | Start value for packet numbers. The number is automatically incremented in the program. |
| clientIdentifier | String [23] | Unique name of the client. This name identifies the client to the broker when the connection is established.<br>The following is permitted:<br>• Numbers<br>• Uppercase and lowercase letters: |
| willTopic | String [100] | If the will-flag is set, then the topic for the last will must be defined here. |
| willMessage | String [100] | If the will-flag is set, then the message for the last will must be defined here. |
| userName | String [20] | If the password flag is set, the username for the login at the broker must be defined here. |
| password | String [20] | If the password flag is set, the password for the login at the broker must be defined here. |
| topic | String [100] | Name for the topic |
| message | String | Message text |

**Note**

Note the following regulations:

1. If you set the "will" flag to "true", you must set a string for the "willMessage" and "willTopic" variables.
2. If you set the "will" flag to false, you must also set the following flags to false:
   - "willQoS_1"
   - "willQoS_2"
   - "willRetain"
3. If you set the flags "username" and "password" to "true", you must store a string with the login data for the variables "userName" and "password". This login data must match the login data that you have stored with the MQTT broker.

## 2.2 Integration into the User project

**Creating a TIA portal project:**

Create a TIA Portal project with the CPU that you want to use for the application example. Parameterize the Ethernet interface of the CPU with an IP address that lies in the same subnet as the MQTT broker.

Connect the SIMATIC controller and the MQTT broker via Ethernet

**Copying the blocks**

The blocks "LMqtt_Publish" and "LMqtt_Data" as well as the required data types are available in the library "LMqtt".

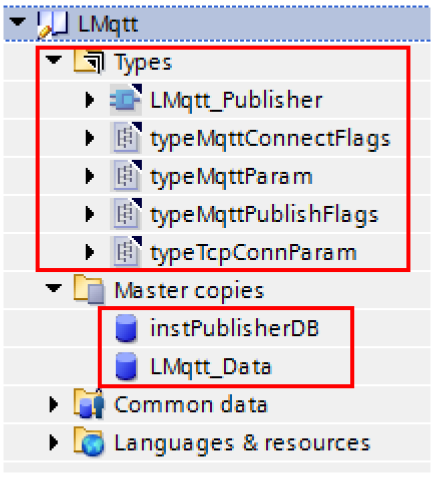
To copy the blocks into your TIA project, follow these instructions:

1. Extract the ZIP file from the download area of this application example (see \ 1 \ in Chapter 4.2) to a local directory on your PC.

2. Open the library view in the TIA Portal. On the toolbar of the "Global library" palette, click the "Open global library" icon.
   The "Open global library" dialog is opened.

3. Navigate to your directory and select the global library "LMqtt". Click on "Open".

4. Copy the contents of "Types" and "Master copies" into your project:



```
LMqtt
  Types
    LMqtt_Publisher
    typeMqttConnectFlags
    typeMqttParam
    typeMqttPublishFlags
    typeTcpConnParam
  Master copies
    instPublisherDB
    LMqtt_Data
  Common data
  Languages & resources
```

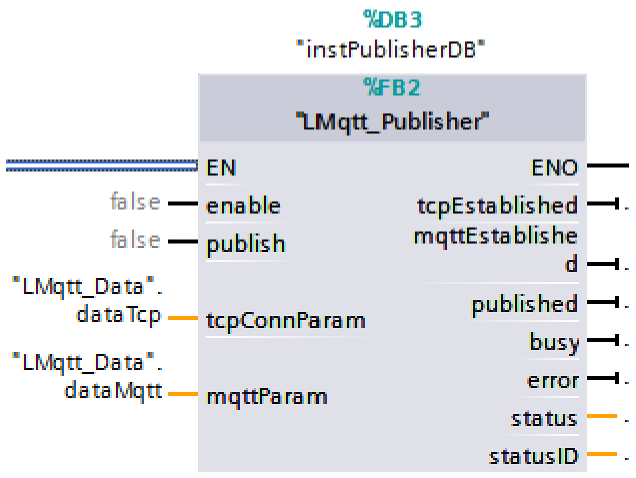
**Call the function block and interconnect**

If you have integrated the blocks into your project, you must still call and interconnect the function block in your program.

1. Call the function module "LMqtt_Publisher" e.g. in OB 1 and assign an instance data block to it.

2. Interconnect the input or output variables as required. Only the interconnection of the input and output variables is specified:

- Input and output variable "tcpConnParam" with "LMqtt_Data".dataTCP
- Input and output variable "mqttParam" with "LMqtt_Data".dataMqtt



## 2.3 Parameter assignment and operation

### Setting the parameters

Before you can test the application example, you must set the parameters for the TCP connection and for MQTT according to your specifications.

All parameters that you can define yourself are in the "LMqtt-Data" data block. Set the parameters in the "Start value" column.

Above all, you must enter your own value for the following parameters:

- IPv4 address or qualified domain name of the MQTT broker. The qualified domain name must ends with a ".".
- remote port on which the MQTT broker receives the messages
- all MQTT parameters, e.g.
  - Flags for the connection
  - Flags for sending messages
  - Logon information at the broker
  - Topic
  - Message text

Then load the project into your CPU.

**Note** If you use the library "LMqttQdn", then you must configure a DNS server in the CPU.

## Operating the application example

Once you have set all the parameters, you can test the application example.

Before you test the application example, check the following points:

1. The project is loaded into the CPU.
2. The CPU and the MQTT broker are connected to each other and can be reached via Ethernet.
3. The MQTT broker is properly configured and started.
4. Logging on to the MQTT broker is started as needed to support the logon of the MQTT client and the publish mechanism.

If the above points are met, you can initiate MQTT communication between the CPU and the MQTT broker. Set the variable "enable" on the function block "LMqtt_Publisher" to the signal "1".

In the positive case, the internal state machines will loop through and establish a TCP or MQTT connection to the MQTT broker. The output variables "tcpConnected" and "mqttConnected" are set and signal an existing TCP or MQTT connection.

Now you can send an MQTT message. To do this, trigger the input variable "publish".

If the connection to the MQTT broker is not established, check the output variables "status" and "statusID" to diagnose the error. The meaning of the values of the two variables can be found in Chapter 2.4.

## 2.4 Error handling

If an error occurs in the program, the current status of the state machines and the cause of the error are written in the output parameters "statusID" and "status".

**"statusID"**

The number of the state in which the error occurred is output at the output "statusID". The states are numbered have the following meanings.

Table 2-8

| Value | Description |
|---|---|
| -12 | MQTT_ERROR |
| -11 | MQTT_DISCONNECTED |
| -2 | TCP_ERROR |
| -1 | TCP_DISCONNECT |
| 0 | IDLE |
| 1 | TCP_PARAM |
| 2 | TCP_CONNECTING |
| 3 | TCP_CONNECTED |
| 10 | MQTT_CONNECT_FLAG_CHECK |
| 11 | MQTT_CONNECT |
| 12 | MQTT_CONNACK |
| 13 | MQTT_PUBLISH |
| 14 | MQTT_PUBACK |
| 15 | MQTT_DISCONNECT |
| 16 | MQTT_CONNECTED |
| 17 | MQTT_PING |
| 18 | MQTT_PINGRESP |
| 19 | MQTT_PUBREL |
| 20 | MQTT_PUBCOMP |
| 20 | TIME_MONITORING |

## "status"

The output parameter "status" displays the error code:

Table 2-9

| statusID | status | Description | Remedy |
|---|---|---|---|
| -1 | Status message from the "TDISCON" block | See manual | - |
| 2 | Status message from the "TCON" block | See manual | Check the availability of the broker.<br>IP address, port, firewall. |
| 3 | Status message from the "TRCV" block | See manual | Check network connection |
| 10 | W#16#80F0 | Error on "Will" flag | Flags of data type "typeMqttConnectFlags" check; |
|  | W#16#80F1 | Error on "WillQoS" flag |  |
|  | W#16#80F3 | Error on "KeepAlive" flag | KeepAlive must exceed 2 seconds. |
| 11 | Status message from the "TSEND" block | See manual | - |
| 12 | 1 | The broker does not accept the MQTT protocol level | Check access data in data type "typeMqttParam" |
|  | 2 | ClientIdentifier is not accepted. |  |
|  | 3 | MQTT service not available |  |
|  | 4 | Data in the username/password are incorrect |  |
|  | 5 | Client is not authorized |  |
| 13 | Status message from the "TSEND" block | See manual | - |
| 14,19,20 | W#16#80F2 | Wrong PacketIdentifier received | - |
| 20 | Status message from the "TCON" or "TRCV" block | Timeout | Check connection parameters |

# 3 Useful information

## 3.1 Basics of MQTT

**Note** A detailed description of MQTT can be found in the MQTT specification description (see \ 3 \ in Chapter 4.2).

### 3.1.1 Terminology

The most important terms in the MQTT telemetry protocol are explained below.

**MQTT message**

A message with MQTT consists of several parts:

- A defined subject ("Topic")
- An assigned "Quality of Service" feature
- The message text

**MQTT client**

An MQTT client is a program or device that uses MQTT. A client always actively establishes the connection to the broker. A client can perform the following functions:

- Send messages with a defined subject ("Topic") in which other clients might be interested to the broker (Publish mechanism)
- Subscribe messages which follow a certain topic (Subscriber mechanism) at the broker
- Unsubscribe yourself from subscribed messages
- Disconnect from the broker

**Note** The function module "LMqtt_Publisher" in this application example supports the following functions:

- Publish mechanism
- Unsubscribe from the broker.

**MQTT broker**

An MQTT broker is the central component of MQTT and can be a program or a device. The broker acts as an intermediary between the sending MQTT client and the subscribing MQTT client. The MQTT broker manages the topics including the messages contained therein and regulates the access to the topics. The broker has the following functions:

- Accept network connections from the clients
- Receive messages from an MQTT client
- Edit subscription requests from MQTT clients
- Forward messages to the MQTT clients that match your subscription

**Note** The MQTT broker is not part of this application example and is assumed to be given.

### Topics

MQTT messages are organized in topics. A topic "describes" a subject area. The topics can be subscribed to by the MQTT clients (subscriber mechanism). The sender of a message (publisher mechanism) is responsible for defining content and topic when sending the message. The broker then takes care that the subscribers get the news from the subscribed topics. The topics follow a defined scheme. They are similar to a directory path and represent a hierarchy.

## 3.1.2 Standard and architecture

### ISO standard

MQTT defines an OASIS or ISO standard (ISO/IEC PRF 20922).

Depending on the security protocols used, MQTT runs on different access ports. Ports offered are:

- 1883: MQTT, unencrypted
- 8883: MQTT, encrypted
- 8884: MQTT, encrypted, client certificate required
- 8080: MQTT via WebSockets, unencrypted
- 8081: MQTT via WebSockets, encrypted

### Architecture

The MQTT is a publish and subscribe protocol. This mechanism decouples a client sending messages (publishers) from one or more clients receiving the messages (subscribers). This also means that the "publishers" know nothing about the existence of the "subscribers" (and vice versa).
There is a third component in the MQTT architecture, the MQTT broker. The MQTT broker is located between "publisher" and "subscriber". The MQTT broker controls the communication.

### 3.1.3 Features

MQTT offers quite useful features.

#### Quality of Service

The MQTT specification provides three levels for message transmission quality assurance:

- QoS "0": The lowest level 0 is a "fire'n'forget" method. This means that there is no guarantee that the message will arrive at all.
- QoS "1": The QoS level 1 ensures that the message ends up in the topic queue at least once. The broker acknowledges receipt of the message.
- QoS "2": In the highest level 2, the broker guarantees by multiple handshake with the client that the message is exactly filed once.

#### Last will

MQTT supports the "Last Will and Testament" feature. This feature is used to notify other clients if the connection to a client has been disconnected accidentally.

Each client can specify its last will while connecting to the broker and notify the broker. This last will is built like a normal MQTT message, including topic, QoS and payload. The broker saves the last will. As soon as the broker notices that the connection with the client in question has been abruptly terminated, the broker sends the last will as an MQTT message to all subscribers who have registered for the topic. In this way, the subscribers also learn that the client has been disconnected.

#### KeepAlive

MQTT supports the KeepAlive feature. The KeepAlive feature ensures that the connection is still open and the client and broker are connected.

For the KeepAlive, the clients define a time interval and communicate it to the broker during their connection setup. This interval is the largest possible tolerated time period in which the client and the broker may remain without contact. If the time is exceeded, the broker must disconnect.

That means that, as long as the client periodically sends messages to the broker within the KeepAlive interval, the client does not need to take any special action to maintain the connection. However, if the client does not send any messages within the KeepAlive interval, they must ping the broker before the deadline expires. With this ping, the client signals to the broker that it is still available.

When a message or a ping packet has been sent to the broker, timing for the KeepAlive interval begins again.

**Note**

- The client determines the KeepAlive interval. It can therefore adjust the interval of his environment, e.g. because of a slow bandwidth.
- The maximum value for the KeepAlive interval is 18 h 12 m 15 s
- When the client sets the KeepAlive interval to "0", the KeepAlive mechanism is disabled.

**Message persistence**

If the connection to a client is interrupted, the broker can cache new messages for this client for later delivery.

**Retained messages**

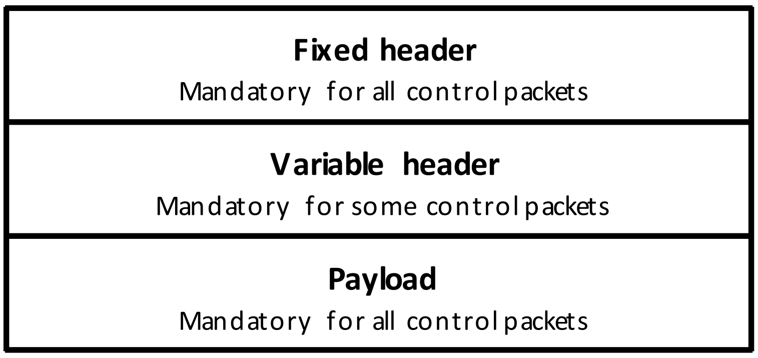
The first time an MQTT client subscribes to a topic, it usually gets a message only when another MQTT client sends a message with the subscribed topic the next time. With "Retained messages", the subscriber receives the last value sent to the topic prior to its subscription request, delivered immediately.

### 3.1.4 MQTT control packets

Most MQTT control packages work according to the handshake procedure. The MQTT client is always the active element and places an order with the broker. The broker confirms the request depending on the order.

The structure of an MQTT control packet is fixed. The following diagram shows the structure.

Figure 3-1


**Fixed header**
Mandatory for all control packets
**Variable header**
Mandatory for some control packets
**Payload**
Mandatory for all control packets

The "Fixed header" always consists of the following elements:

- An identifier number for the MQTT control packet type
- An area for possible flags; if no flags are provided for the control packet, the bits are marked as "reserved"
- The number of following bytes after the "Fixed header"

The "Variable header" is required only for some control packages. The content of the variable header depends on the control packet type.

The payload is mandatory for most control packets. Again, the content depends on the control packet type. For each type of control packet, there are clear rules with what and in what order the payload can be filled.

**Note** A detailed description of MQTT control packets can be found in the MQTT specification description (see \ 3 \ in Chapter 4.2).

The MQTT control packets from this application example are briefly explained below.

#### MQTT Connection

An MQTT connection is always made between a client and the broker. A direct client-client connection is not possible.

The connection is initiated by a client as soon as the client sends a "CONNECT" packet to the broker. If positive, the broker replies with a "CONNACK" packet and a status code.

The broker immediately closes the connection in the following cases:

- If the "CONNECT" packet is faulty
- If the structure of the "CONNECT" packet does not meet the specification
- If the connection takes too long

A "CONNECT" packet contains an area for flags in the "Variable Header". The "CONNECT" flag byte contains a number of parameters that specify the behavior of the MQTT connection. In addition, the "CONNECT" flag byte also shows which optional fields are present in the "payload" or not.

The following fields are mandatory in the "payload":

- The "ClientID" is used to identify the client at the broker
- The connection type can be regulated with the "CleanSession"
- The KeepAlive time determines the time interval in which the client is obligated to report to the broker. This can be done either by sending a message or a PING command. If the client does not report in the time interval, the broker disconnects from the client.

Examples of optional fields are username, password and information about the last will.

**MQTT-push mechanism**

Once an MQTT client connects to the broker, it can send messages to the broker. To do this, the client uses the "PUBLISH" packet. Because MQTT messages are filtered and managed based on topics, each MQTT message must contain a topic. The topic is part of the "Variable Header". The actual message text is contained in the "payload".

Depending on the quality assurance setting ("QoS"), the push mechanism ends at this point or other control packets are exchanged:

If QoS is equal to "0", the send job ends here.

With QoS equal to "1", the broker acknowledges the "PUBLISH" packet with a "PUBACK".

With QoS equal to "2", the broker acknowledges the "PUBLISH" packet with a "PUBREC". This is followed by another handshake between client and broker. The client answers the "PUBREC" with a "PUBREL" packet. The broker completes the double handshake with a "PUBCOM" packet.

**Note** You can find further information on Quality Assurance QoS in Chapter3.1.3.

**MQTT-ping mechanism**

If the KeepAlive function is active (the KeepAlive interval is greater than "0"), the client must send at least one message to the broker within the KeepAlive interval. If this is not the case, the broker must terminate the connection to the client. To prevent this type of forced abort, the client must ping the broker before the KeepAlive time expires. The control packet "PINGREQ" is used for this. The broker responds with a "PINGRESP" packet and signals its availability to the client.

**Note** This application example assumes an active KeepAlive function. The KeepAlive interval must be greater than two seconds.

**MQTT disconnection**

A client can close the connection to a broker by sending a "DISCONNECT" packet to the broker. The broker then deletes all "Last Will and Testament" information. As the client is actively and voluntarily connected, the broker does not send its last wishes to the registered subscribers.

## 3.2 How the LMqtt_Publisher FB works

### 3.2.1 Requirements and implementation

The following conditions must be fulfilled for a communication relationship between an MQTT client and an MQTT broker:

1. A TCP connection to the MQTT broker has been successfully established (status: "TCP_CONNECTED").
2. The function block "LMqtt_Publisher" has logged on to the broker via the existing TCP connection as an MQTT client and has connected to it (status: "MQTT_CONNECTED").
3. The trigger to send the message or to receive the MQTT connection ("KeepAlive") is active. Depending on the desired quality assurance, the message is sent to the broker via the existing MQTT connection.

**Note** An MQTT connection setup is only possible if the TCP connection to the broker is successfully established and then maintained.

An MQTT message or KeepAlive can only be sent if there is a TCP and MQTT connection to the broker.

**Overview**

To fulfill the mentioned requirements, several state machines were realized in the program:

- State machine "TCP": Management of the TCP connection
- State machine "MQTT": Management of the MQTT connection
- State machine "PUSH": Handling of the transfer

### 3.2.2 State machine "TCP"

The state machine "TCP" is started if a positive edge was detected at the input parameter "enable". This state machine has the following functions:

- It controls the structure of the TCP connection
- It monitors the existing TCP connection for connection errors, e.g. cable breakage
- If an error has occurred or no positive edge was detected at the "enable" input parameter, it sets all static variables and the other state machines to a defined state.

The state machine "TCP" contains the following states:

- IDLE
- TCP_PARAM
- TCP_CONNECTING
- TCP_CONNECTED
- TCP_DISCONNECT
- TCP_ERROR

The meaning of the states is listed in the following table

Table 3-1

| Status | Description |
|---|---|
| IDLE | In "IDLE" state, all parameters are reset.<br>The state machine waits in this state until it detects a positive edge at the input parameter "enable". As soon as a positive edge is applied to the input, the state machine is set to the "TCP_PARAM" state. |
| TCP_PARAM | All connection parameters are read in this state. The function block changes to the state "TCP_CONNECTING" without a switching condition. |
| TCP_CONNECTING | The TCP connection to the MQTT broker is established in this state. If the connection with "TCON" has been established successfully, the FB changes to the "TCP_CONNECTED" state and the output variable "tcpConnected" is set. The TCP connection persists until it is terminated with "TDISCON".<br>If an error occurs during connection setup, the state machine switches to the "TCP_ERROR" state. |
| TCP_CONNECTED | In this state, the function module maintains the state until the following events occur:<br>• The "TRCV" block detects a connection abort, e.g. by the network cable being pulled out, and reports an error.<br>• The input parameter "enable" is reset and thus initiates the disconnection.<br>If the "TRCV" block detects an error, the state machine changes to the "TCP_ERROR" state.<br>The "TCP_CONNECTED" state is a prerequisite for the processing of the state machine "MQTT". |
| TCP_DISCONNECT | The TCP connection is disconnected in this state. If the "TDISCON" block detects an error, the state machine changes to the "TCP_ERROR" state. |
| TCP_ERROR | If an error occurs in the state machine "TCP", the state "TCP_ERROR" is the central point of contact. Here, the required parameters (static variables and output variables) are set or reset and the MQTT connection is aborted. In addition, the following actions are carried out:<br>• The error message of the T-block involved is transferred at the output "status".<br>• The number of the state in which the error occurred is output at the output "statusID"<br>• The state machine returns to the "IDLE" state. If there is already a TCP connection, it will be disconnected in advance. The output variable "tcpConnected" is reset.<br>• The state machine "MQTT" is set to the state "MQTT_DISCONNECTED". |

**Note** The function block "LMqtt_Publisher" is not "self-healing" in the event of an error. This means that the function block falls back into the "IDLE" state and remains there until a new positive edge is detected at the "enable" input parameter.

### 3.2.3 State machine "MQTT"

The state machine "MQTT" is automatically started when the state machine "TCP" reaches the state "TCP_CONNECTED". This state machine has the following functions:

- It controls the handshake procedure for setting up the MQTT connection
- It ensures the disconnection
- It manages the internal state machine "PUSH" to send messages
- It makes sure that a PING packet is sent before the KeepAlive interval expires.

The state machine "MQTT" contains the following states

- MQTT_DISCONNECTED
- MQTT_CONNECT_FLAG_CHECK
- MQTT_CONNECT
- MQTT_CONNACK
- MQTT_CONNECTED
- MQTT_DISCONNECT
- MQTT_ERROR

The meaning of the states is listed in the following table:

Table 3-2

| Status | Description |
|---|---|
| MQTT_DISCONNECTED | As long as there is no TCP connection, the state is always "MQTT_DISCONNECTED".<br>Only when a TCP connection has been established is the switching condition automatically activated for the status "MQTT_CONNECT_FLAG_CHECK". |
| MQTT_CONNECT_FLAG_CHECK | n this state, the flags and parameters for the MQTT connection setup are read in and validated. If there are discrepancies during the check, the state machine changes to the state "MQTT_ERROR" and a corresponding error message is output at the output parameter "status". In the error-free state, the state machine switches to the state "MQTT_CONNECT" without a switching condition. |
| MQTT_CONNECT | The MQTT connection to the MQTT broker is established in this state. For this a "CONNECT" packet with the read in parameters is assembled and sent to the broker with the "TSEND" block.<br>If an error occurs while sending the "CONNECT" packet, the state machine will change to the "MQTT_ERROR" state. If the "CONNECT" packet has been sent successfully, the state machine changes to the "MQTT_CONNACK" state. |
| MQTT_CONNACK | The state machine maintains this state until the "TRCV" block receives a message. It is checked whether it is a "CONNACK" packet. If the broker has confirmed the connection request with "CONNACK", the state machine changes to the state "MQTT_CONNECTED" and the output variable "mqttConnected" is set. The KeepAlive interval is started if necessary.<br>If the "TRCV" block detects an error, the state machine changes to the "MQTT_ERROR" state. |
| MQTT_CONNECTED | In this state, the function module maintains the state until the MQTT connection or TCP connection is cleared. In the "MQTT_CONNECTED" state, the following points are checked cyclically: |

| Status | Description |
|---|---|
| | • Is there a send impulse for an MQTT message?<br>• Will the KeepAlive interval soon end and a PING command have to be sent to the broker?<br>Depending on the outcome of the check, the internal state machine "PUSH" is set to the appropriate state to execute the desired routine. |
| MQTT_DISCONNECT | If the input parameter "enable" is reset, the MQTT connection is cleared. For this a "DISCONNECT" packet is assembled and sent with the "TSEND" block to the broker.<br>If an error occurs while sending the "DISCONNECT" packet, the state machine will change to the "MQTT_ERROR" state. If the "DISCONNECT" packet has been sent successfully, the state machine changes to the "MQTT_DISCONNECTED" state. At the same time, the state machine "TCP" is set to the "TCP_DISCONNECT" state. This also ends the TCP connection. |
| MQTT_ERROR | If an error occurs in the state machine "MQTT", the state "MQTT_ERROR" is the central point of contact. Here, the required parameters (static variables and output variables) are set or reset. In addition, the following actions are carried out:<br>• The error message of the MQTT command involved is transferred at the output "status".<br>• The number of the state in which the error occurred is output at the output "statusID"<br>• The state machine returns to the "MQTT_DISCONNECTED" state. |

### 3.2.4 State machine "PUSH"

The state machine "PUSH" is only run through when the state machine "MQTT" is in the "MQTT_CONNECTED" state. This is because it is decided here from which point the state machine "PUSH" is started. If there is a send impulse for a MQTT message, then the send routine becomes active. If the KeepAlive time is ending soon, the PING routine starts.

The state machine "PUSH" contains the following states:

- IDLE
- MQTT_PUBLISH
- MQTT_PUBACK
- MQTT_PUBREL
- MQTT_PUBCOMP
- MQTT_PING
- MQTT_PINGRESP

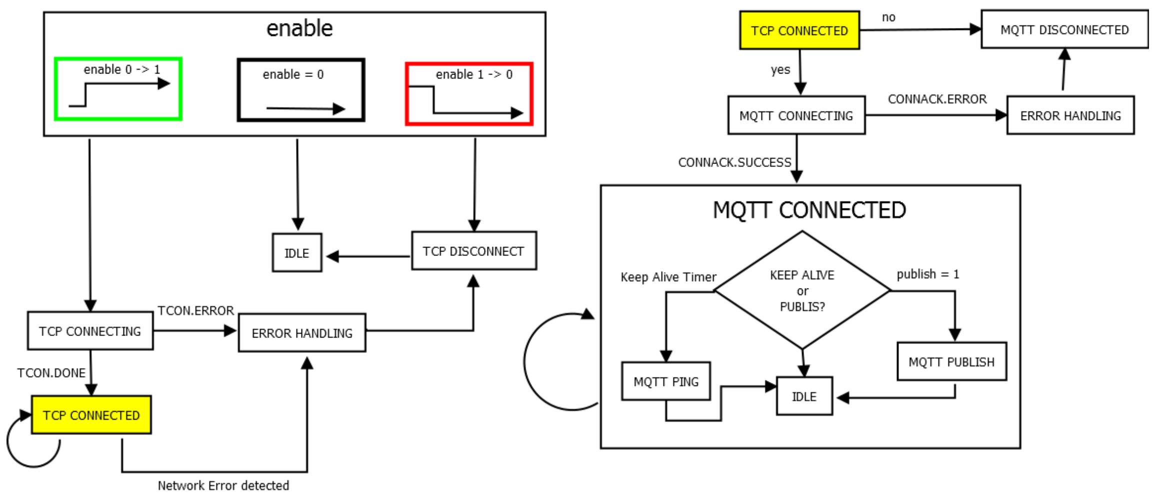
| Status | Description |
|---|---|
| IDLE | As long as there is no transmission impulse or the KeepAlive interval is not expiring, the state is always "IDLE". |
| MQTT_PUBLISH | If a positive edge was detected at input parameter "publish" in state "MQTT_CONNECTED", the internal state machine "PUSH" is set to state "MQTT_PUBLISH". The transmission routine starts here depending on the quality assurance QoS.<br>First, a "PUBLISH" packet with the given parameters, the topic and the message text is assembled and then it is sent to the broker with the "TSEND" block.<br>If an error occurs while sending the "PUBLISH" packet, the state machine "MQTT" changes to state "MQTT_ERROR" and the state machine goes back to "IDLE".<br>If the "PUBLISH" packet has been sent successfully, the next step depends on the selected QoS:<br>• If QoS is equal to "0", the transmission process ends here and this state machine returns to "IDLE". The KeepAlive interval is restarted if necessary.<br>• With QoS equal to "1" and QoS equal to "2", this state machine changes to the state "MQTT_PUBACK" to receive an acknowledgment from the broker. |
| MQTT_PUBACK | If the QoS is greater than "0", the client expects the broker to be acknowledged on the "PUBLISH" packet.<br>The state machine maintains this state until the "TRCV" block receives a message. It is checked whether it is a "PUBACK" packet.<br>If the broker has confirmed receipt of the message, the next step depends on the chosen QoS:<br>• If QoS is equal to "1", the transmission process ends here and this state machine returns to "IDLE". The KeepAlive interval is restarted if necessary.<br>• If QoS is equal to "2", this state machine changes to the state "MQTT_PUBREL to confirm the acknowledgment input.<br>If the "TRCV" block detects an error, the state machine changes to the "MQTT_ERROR" state and the state machine returns to "IDLE". |
| MQTT_PUBREL | If QoS is equal to "2", there will be a double handshake with the broker. |

| Status | Description |
|---|---|
| | After the client has received the "PUBACK", it is confirmed by the "PUBREL" packet. For this purpose, a "PUBREL" packet is assembled and then sent to the broker with the "TSEND" block. If an error occurs while sending the "PUBREL" packet, the state machine "MQTT" changes to state "MQTT_ERROR" and the state machine goes back to "IDLE".<br>If the "PUBREL" packet has been sent successfully, the state machine changes to the "PUBCOMP" state. |
| MQTT_PUBCOMP | This state is the last part of the dual handshake procedure at QoS equal to "2". The client expects the broker to acknowledge the "PUBREL" packet.<br>The state machine maintains this state until the "TRCV" block receives a message. It is checked whether it is a "PUBCOMP" packet.<br>If the broker has acknowledged receipt of the message, this state machine will return to IDLE and the KeepAlive interval will be restarted if necessary. The handshake procedure is now complete. If the "TRCV" block detects an error, the state machine changes to the "MQTT_ERROR" state and this state machine returns to "IDLE". |
| MQTT_PING | If it is determined in the "MQTT_CONNECTED" state that the KeepAlive interval is expiring, the internal state machine is set to the "MQTT_PING" state. This is where the ping routine starts.<br>First a "PING" packet is assembled and then sent to the broker with the "TSEND" block.<br>If an error occurs while sending the "PING" packet, the state machine "MQTT" changes to state "MQTT_ERROR" and this state machine goes back to "IDLE". |
| MQTT_PINGRESP | After the "PING" packet, the client expects the broker to be acknowledged.<br>The state machine maintains this state until the "TRCV" block receives a message. It is checked whether it is a "PINGRESP" packet.<br>When the broker has confirmed receipt of the message, the state machine goes back to "IDLE". The KeepAlive interval is restarted.<br>If the "TRCV" block detects an error, the state machine changes to the "MQTT_ERROR" state and this state machine returns to "IDLE". |

### 3.2.5 Function diagram

The following figure shows the diagram of the operation with the three state machines:

Fig. 3-2



# 4 Appendix

## 4.1 Service and Support

### Industry Online Support

Do you have any questions or need assistance?

Siemens Industry Online Support offers round the clock access to our entire service and support know-how and portfolio.

The Industry Online Support is the central address for information about our products, solutions and services.

Product information, manuals, downloads, FAQs, application examples and videos – all information is accessible with just a few mouse clicks:
https://support.industry.siemens.com

### Technical Support

The Technical Support of Siemens Industry provides you fast and competent support regarding all technical queries with numerous tailor-made offers – ranging from basic support to individual support contracts. Please send queries to Technical Support via Web form:
www.siemens.com/industry/supportrequest

### SITRAIN – Training for Industry

We support you with our globally available training courses for industry with practical experience, innovative learning methods and a concept that's tailored to the customer's specific needs.

For more information on our offered trainings and courses, as well as their locations and dates, refer to our web page:
www.siemens.com/sitrain

### Service offer

Our range of services includes the following:

- Plant data services
- Spare parts services
- Repair services
- On-site and maintenance services
- Retrofitting and modernization services
- Service programs and contracts

You can find detailed information on our range of services in the service catalog web page:
https://support.industry.siemens.com/cs/sc

### Industry Online Support app

You will receive optimum support wherever you are with the "Siemens Industry Online Support" app. The app is available for Apple iOS, Android and Windows Phone:
https://support.industry.siemens.com/cs/ww/en/sc/2067

## 4.2 Links and Literature

Table 4-1

| No. | Topic |
|---|---|
| \1\ | Siemens Industry Online Support<br>https://support.industry.siemens.com |
| \2\ | Link to this entry<br>https://support.industry.siemens.com/cs/ww/en/view/109748872 |
| \3\ | MQTT specification<br>http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html |
| \4\ | Information on local_device_id<br>https://support.industry.siemens.com/cs/ww/en/view/51339682 |

## 4.3 Change documentation

Table 4-2

| Version | Date | Modifications |
|---|---|---|
| V1.0 | 03/2018 | First version |
| V1.1 | 08/2018 | Insert Library "LMqttQdn" for SIMATIC S7-1500 |