MQTT client for SIMATIC S7-1500 and S7-1200

Blocks for S7-1500 and S7-1200

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

Siemens Industry Online Support

Legal information

Use of application examples

Application examples illustrate the solution of automation tasks through an interaction of several components in the form of text, graphics and/or software modules. The application examples are a free service by Siemens AG and/or a subsidiary of Siemens AG ("Siemens"). They are non-binding and make no claim to completeness or functionality regarding configuration and equipment. The application examples merely offer help with typical tasks; they do not constitute customer-specific solutions. You yourself are responsible for the proper and safe operation of the products in accordance with applicable regulations and must also check the function of the respective application example and customize it for your system.
Siemens grants you the non-exclusive, non-sublicensable and non-transferable right to have the application examples used by technically trained personnel. Any change to the application examples is your responsibility. Sharing the application examples with third parties or copying the application examples or excerpts thereof is permitted only in combination with your own products. The application examples are not required to undergo the customary tests and quality inspections of a chargeable product; they may have functional and performance defects as well as errors. It is your responsibility to use them in such a manner that any malfunctions that may occur do not result in property damage or injury to persons.

Disclaimer of liability

Siemens shall not assume any liability, for any legal reason whatsoever, including, without limitation, liability for the usability, availability, completeness and freedom from defects of the application examples as well as for related information, configuration and performance data and any damage caused thereby. This shall not apply in cases of mandatory liability, for example under the German Product Liability Act, or in cases of intent, gross negligence, or culpable loss of life, bodily injury or damage to health, non-compliance with a guarantee, fraudulent non-disclosure of a defect, or culpable breach of material contractual obligations. Claims for damages arising from a breach of material contractual obligations shall however be limited to the foreseeable damage typical of the type of agreement, unless liability arises from intent or gross negligence or is based on loss of life, bodily injury or damage to health. The foregoing provisions do not imply any change in the burden of proof to your detriment. You shall indemnify Siemens against existing or future claims of third parties in this connection except where Siemens is mandatorily liable.
By using the application examples you acknowledge that Siemens cannot be held liable for any damage beyond the liability provisions described.

Other information

Siemens reserves the right to make changes to the application examples at any time without notice. In case of discrepancies between the suggestions in the application examples and other Siemens publications such as catalogs, the content of the other documentation shall have precedence.
The Siemens terms of use (https://support.industry.siemens.com) shall also apply.

Security information

Siemens provides products and solutions with Industrial Security functions that support the secure operation of plants, systems, machines and networks.
In order to protect plants, systems, machines and networks against cyber threats, it is necessary to implement – and continuously maintain – a holistic, state-of-the-art industrial security concept. Siemens' products and solutions constitute one element of such a concept.
Customers are responsible for preventing unauthorized access to their plants, systems, machines and networks. Such systems, machines and components should only be connected to an enterprise network or the Internet if and to the extent such a connection is necessary and only when appropriate security measures (e.g. firewalls and/or network segmentation) are in place.
For additional information on industrial security measures that may be implemented, please visit https://www.siemens.com/industrialsecurity.
Siemens' products and solutions undergo continuous development to make them more secure. Siemens strongly recommends that product updates are applied as soon as they are available and that the latest product versions are used. Use of product versions that are no longer supported, and failure to apply the latest updates may increase customer's exposure to cyber threats.
To stay informed about product updates, subscribe to the Siemens Industrial Security RSS Feed at: https://www.siemens.com/industrialsecurity.

Table of contents

Legal information **2**

1 Introduction **4**

1.1 Overview 4
1.2 Principle of operation 6
1.3 Components used 7

2 Engineering **9**

2.1 Block description 9
2.1.1 Interface description "LMQTT_Client" 9
2.1.2 Overview of data types 11
LMQTT_typePublishData 15
2.2 Configuration 17
2.2.1 Create TIA Portal Project 17
2.3 Integration of the function block in the user program 18
2.3.1 Opening the global library "LMQTT_Client" 18
2.3.2 Copying function blocks and data types to the user program 20
2.3.3 Creating global data block 21
2.3.4 Calling function blocks in the user program 24
2.4 Configuration of the security feature 26
2.4.1 Using the global certificate manager in the TIA Portal 27
2.4.2 Using the local certificate manager of the CPU 32
2.5 Parameter assignment and operation 35
2.6 Error handling 37

3 Useful information **41**

3.1 Basics of MQTT 41
3.1.1 Terminology 41
3.1.2 Standard and architecture 42
3.1.3 Features 43
3.1.4 Structure of the MQTT control packets 45
3.1.5 MQTT connection 46
3.1.6 MQTT-push mechanism 49
3.1.7 MQTT sub-mechanism 52
3.1.8 MQTT-ping mechanism 55
3.1.9 MQTT disconnection 56
3.2 How the "LMQTT_Client" FB works 57
3.2.1 Requirements and implementation 57
3.2.2 State machine "TCP state machine": 57
3.2.3 State machine "MQTT state machine": 59
3.2.4 State machine "MQTT job state machine": 62
3.2.5 Function diagram 68

4 Appendix **69**

4.1 Service and support 69
4.2 Links and Literature 70
4.3 Change documentation 70

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:

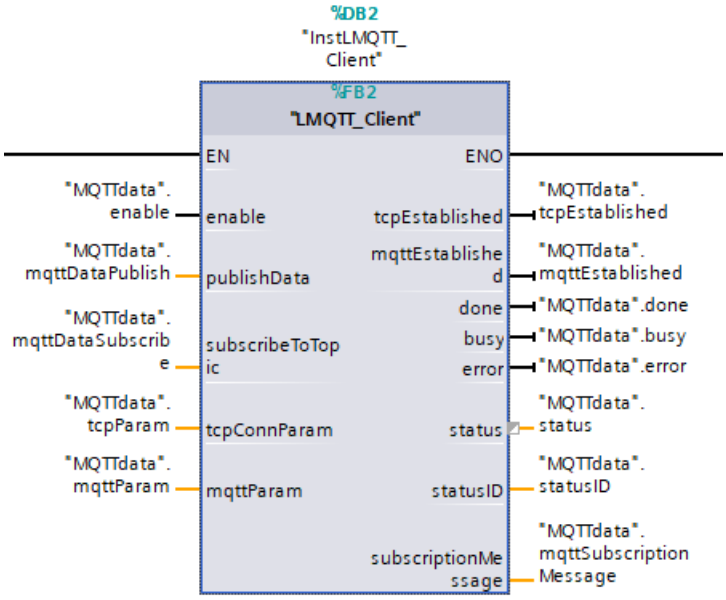
- Lightweight protocol with low transport overhead
- Minimal need for network bandwidth through push mechanism
- Reconnect function after termination of connection
- Resending of messages after disconnection
- Mechanism for notifying prospects of an unforeseen disconnection of a client
- Easy to use 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

The "LMQTT_Client" library offers you an adequate solution for implementing the MQTT protocol in a SIMATIC S7 controller.

The "LMQTT_Client" library provides you with one function block each for the SIMATIC S7-1500 and SIMATIC S7-1200. The function block "LMQTT_Client" integrates the MQTT client function and allows you to transfer MQTT messages to a broker (publisher role) and to create subscriptions (subscriber role). The communication can be secured via a TLS connection.

Figure 1-1



Note The MQTT client supports MQTT protocol version 3.1.1.

1.2 Principle of operation

Schematic representation

The following figure shows the most important relationships between the components involved and the steps required for secured MQTT communication (MQTT over TLS).

Figure 1-2

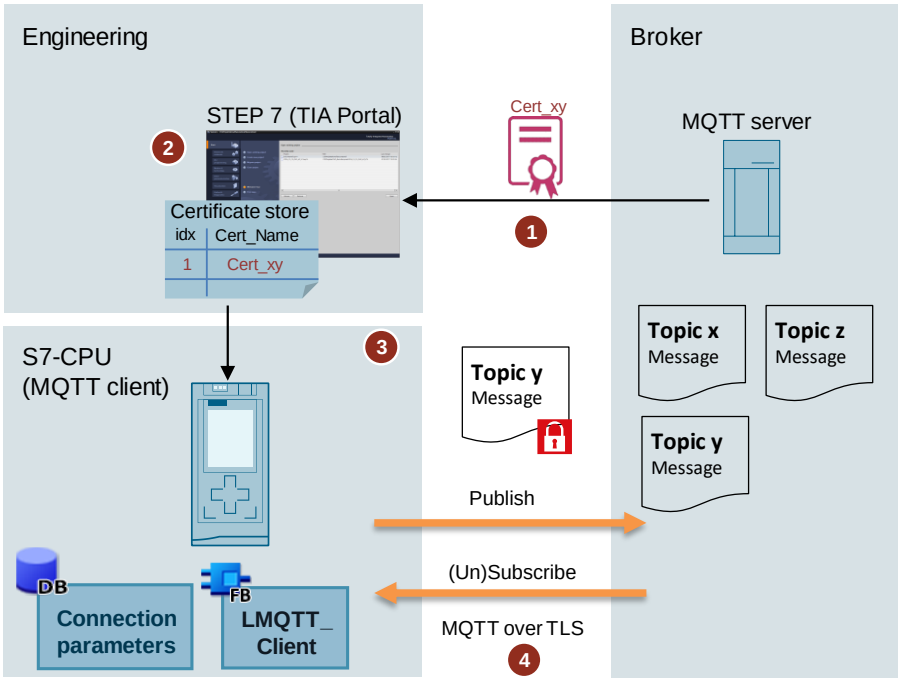


Table 1-1

Step	Description
1	Determine the CA certificate of the MQTT broker.
2	Importing the third-party certificate into STEP 7 (TIA Portal) The certificate is now in the global certificate manager of STEP 7.
3	You must assign the imported certificate to the S7 CPU. To recognize the certificate as valid, the time of the S7-CPU must be current.
4	The function block "LMQTT_Client" assumes the following roles: • Publisher to send MQTT messages to the MQTT broker • Subscriber to subscribe to MQTT messages or end subscriptions The MQTT message is encrypted via a secure connection (MQTT over TLS).

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

1.3 Components used

The following hardware and software components were used to create this application example:

Table 1-2

Components	Quantity	Article number	Note
CPU 1513-1 PN	1	6ES7513-1AL01-0AB0	• Alternatively you can use another S7 1500 CPU or ET 200 CPU (ET 200SP, ET 200pro). At least firmware version 2.0 is required for secure MQTT communication via TLS. • Alternatively you can use an S7-1200 CPU with firmware V4.3 or higher. • Alternatively you can use one of the following components: - CP 1243-1 (6GK7243-1BX30-0XE0) with firmware V3.2 or higher - CP 1243-7 LTE (6GK7243-7KX30-0XE0/ 6GK7243-7SX30-0XE0) with firmware V3.2 or higher
TIA Portal V15.1	-	DVD: 6ES7822-1AA05-0YA5 Download: 6ES7822-1AE05-0YA5	-
TIA Portal V16	-	DVD: 6ES7822-1AA06-0YA5 Download: 6ES7822-1AE06-0YA5	-
MQTT broker	-	-	If you want to encrypt the communication, the MQTT broker must support SSL/TLS.

This application example consists of the following components:

Table 1-3

Components	File name
"LMQTT_Client" library	109748872_MQTT_Client_LIB_V2-0.zip
This document	109748872_MQTT_Client_DOKU_V2-0_de.pdf

Note

With S7-1500 CPUs you can reach the MQTT broker via a static IP address or a domain name ("Qualified Domain Name", in short: QDN) using the "LMQTT_Client" library.

With S7-1200 CPUs you can reach the MQTT broker via a static IP address using the "LMQTT_Client" library.

With S7-1200 CPUs you can reach the MQTT broker with TIA Portal V16 library block and Firmware 4.4 via a domain name ("Qualified Domain Name", in short: QDN).

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_Client"

Note The function block "LMQTT_Client" is available for S7-1200 CPUs and S7-1500 CPUs. The input parameter "tcpConnParam" of the data type "LMQTT_typeTcpConnParamData" is not identical.

The function block "LMQTT_Client" is designed for "optimized block access".

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

Input parameters

Table 2-1

Parameters	Data type	Function
enable	BOOL	Control parameter TRUE: The connection to the MQTT broker is established and maintained. FALSE: The connection is terminated.
publishData	LMQTT_typePublishData	If a connection is established, data can be sent to the MQTT broker (publish). Table 2-7 shows the structure of the data type "LMQTT_typePublishData".
subscribeToTopic	LMQTT_typeSubscribeData	With an existing connection, data can be subscribed to by the MQTT broker. When data is received, it is provided at the "subscriptionMessage" output. Table 2-8 shows the structure of the data type "LMQTT_typeSubscribeData".

Parameters	Data type	Function
tcpConnParam	LMQTT_typeTcpConnParamData	Configuration of the TCP connection parameters to be used when establishing the connection. Table 2-3 and Table 2-4 show the structure of the data type "LMQTT_ typeTcpConnParamData".
mqttParam	LMQTT_typeParamData	Configuration of the MQTT connection parameters to be used when establishing the connection. Table 2-5 and Table 2-6 show the structure of the data type "LMQTT_typeParamData".

Output parameters

Table 2-2

Parameters	Data type	Function
tcpEstablished	BOOL	True if the TCP connection has been established
mqttEstablished	BOOL	True if the MQTT connection has been established
done	BOOL	True for a cycle if one of the following jobs was successfully completed: • Connection successfully established (connect) • Message successfully sent to MQTT broker (publish) • Message successfully received from MQTT broker (subscribe)
busy	BOOL	True, while a PING is sent to the MQTT broker or one of the following jobs is running: • Connect • Message sent to MQTT broker (publish) • Message received from MQTT broker (subscribe)
Error	BOOL	True if there is an error
status	DWORD	Error code
statusID	INT	State that caused the error
subscriptionMessage	LMQTT_typeSubscriptionData	Received message for the subscribed topic (received data) Table 2-9 shows the structure of the data type "LMQTT_typeSubscriptionData".

2.1.2 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:

- "LMQTT_typeTcpConnParamData"
- "LMQTT_typeParamData"
- "LMQTT_typePublishData"
- "LMQTT_typeSubcripeData"

Data type "LMQTT_typeTcpConnParamData" for S7-1500

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

Note If you are using the Library for TIA Portal V16, the description for this parameter also applies on the S7-1200.

Table 2-3

Parameters	Data type	Meaning
useQdn	BOOL	True, if the TCP connection is to be established via the fully qualified domain name.
hwIdentifier	HW_ANY	HW ID of the PROFINET interface of the CPU
connectionID	CONN_OUC	ID of the TCP connection
qdnAdressBroker	String [254]	URL of the MQTT broker The URL must end with a dot (".).
ipAdressBroker	Array [0..3] of USInt	IP address of the MQTT broker, e.g. 192.168.0.10 ipAdressBroker[0] = 192 ipAdressBroker[1] = 168 ipAdressBroker[2] = 0 ipAdressBroker[3] = 10
localPort	UINT	Local port number in the CPU
mqttPort	UINT	Remote port on the MQTT broker
activateSecureConn	BOOL	True, if the communication is to be secured via TLS.
validateSubjectAlternateNameOfServer	BOOL	A set bit means that the TLS client validates the alternative name of the certificate owner. Only relevant when "activateSecureConn" equals "true".
idTlsServerCertificate	UDINT	ID of the X.509 V3 certificate (usually a CA certificate) to validate the TLS server authentication. If this parameter is "0", the TLS client uses all (CA) certificates currently loaded in the client's certificate store to validate server authentication. Only relevant when "activateSecureConn" equals "true".
idTlsOwnCertificate	UDINT	ID of your own X.509 V3 certificate to validate your own authentication against the TLS server. Only relevant when "activateSecureConn" equals "true" and the TLS server requests client authentication.

Data type "LMQTT_typeTcpConnParamData" for S7-1200

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

Table 2-4

Parameters	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 USInt	IP address of the MQTT broker, e.g. 192.168.0.10 ipAdressBroker[0] = 192 ipAdressBroker[1] = 168 ipAdressBroker[2] = 0 ipAdressBroker[3] = 10
localPort	UINT	Local port number in the CPU
mqttPort	UINT	Remote port on the MQTT broker
activateSecureConn	BOOL	True, if the communication is to be secured via TLS.
validateSubjectAlternateNameOfServer	BOOL	A set bit means that the TLS client validates the alternative name of the certificate owner. Only relevant when "activateSecureConn" equals "true".
idTlsServerCertificate	UDINT	ID of the X.509 V3 certificate (usually a CA certificate) to validate the TLS server authentication. If this parameter is "0", the TLS client uses all (CA) certificates currently loaded in the client's certificate store to validate server authentication. Only relevant when "activateSecureConn" equals "true".
idTlsOwnCertificate	UDINT	ID of your own X.509 V3 certificate to validate your own authentication against the TLS server. Only relevant when "activateSecureConn" equals "true" and the TLS server requests client authentication.

Data type "LMQTT_typeParamData"

This data type contains all the information about MQTT, e.g.

- Flags for the connection
- Logon information at the broker

The following table shows the structure of the "LMQTT_typeParamData" data type.

Table 2-5

Parameters	Data type	Meaning
connectFlag	LMQTT_typeConnectFlags	To display the large number of parameters in a more structured way, a separate data type has been created for the flags. With the data type "LMQTT_typeConnectFlags" you can determine the flags for establishing the connection to the MQTT broker. For information on the structure of the data type "LMQTT_typeConnectFlags", see Table 2-6.
keepAlive	UInt	Time interval of the KeepAlive function in seconds. If the parameter keepAlive=0, the KeepAlive function is deactivated.
clientIdentifier	String [23]	Unique name of the client. This name identifies the client to the MQTT broker when the connection is established. The following characters are permitted: • Numbers • 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 [100]	If the password flag is set, the username for the login at the broker must be defined here.
password	String [200]	If the password flag is set, the password for the login at the broker must be defined here.

The following table shows the structure of the "LMQTT_typeConnectFlags" data type.

Table 2-6

Parameters	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.
willQoS1	BOOL	True if the QoS for last will is Level 1.
willQoS2	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.

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.

LMQTT_typePublishData

This data type contains topic, message text, and flags for the MQTT message sent to the MQTT broker.

The following table shows the structure of the "LMQTT_typePublishData" data type.

Table 2-7

Parameters	Data type	Meaning
publishMessage	BOOL	Set the "publishMessage" parameter to TRUE to send data to the MQTT broker. If the data has been successfully transferred to the MQTT broker (output "done" of the FB "LMQTT_Client" is set to "TRUE" for one cycle), the parameter "publishMessage" can be reset to "FALSE".
publishTopic	WString [250]	Name for the topic The size must be adapted to the following constants in the FB: • MAX_SENDBUFFER • MAX_PUBLISH_MESSAGE **Note** The "publishTopic" parameter must not contain an empty string.
publishMessageData	• STEP 7 V15.1 WString [1500] • STEP 7 V16 WString [800]	Message text The size must be adapted to the following constants in the FB: • MAX_SENDBUFFER • MAX_PUBLISH_MESSAGE
publishQoS	Int	Defines the QoS level for the MQTT message. Possible values are: • "0" for QoS level 0 • "1" for QoS level 1 • "2" for QoS level 2
publishRetainFlag	BOOL	True if the message should be saved to the broker.

LMQTT_typeSubscribeData

This data type allows you to subscribe to or unsubscribe from a topic. This allows you to receive data from the MQTT broker.

The following table shows the structure of the "LMQTT_typeSubscribeData" data type.

Table 2-8

Parameters	Data type	Meaning
subscribeToTopic	Bool	True to subscribe to the topic indexed with the "subscriptionTopic" parameter.
unsubscribeFromTopic	Bool	True to unsubscribe from the topic indexed with the "subscriptionTopic" parameter.
subscriptionTopic	WString [250]	Name of the topic to be subscribed to or unsubscribed from by the MQTT broker. **Note** The "subscriptionTopic" parameter must not contain an empty string.
subscriptionQoS	Int	Defines the QoS level for the MQTT message in the subscriber Possible values are: • "0" for QoS level 0 • "1" for QoS level 1

LMQTT_typeSubscriptionData

This data type contains the received data.

The following table shows the structure of the "LMQTT_typeSubscriptionData" data type.

Table 2-9

Parameters	Data type	Meaning
newMessageReceived	BOOL	True for a cycle if data was successfully received.
messageInvalid	BOOL	True if one of the following conditions is met: • Internal buffer of the FB "LMQTT_Client" is too small. • The length of the string "receivingTopic" or "receivedMessageData" is too small.
receivingTopic	WString [400]	Name of the subscribed topic
receivedMessageData	WString [2400]	Received data (text of message for the subscribed topic)

2.2 Configuration

2.2.1 Create TIA Portal Project

1. Create a TIA Portal project with the CPU that you want to use for the application example.
2. Parameterize the Ethernet interface of the CPU with an IP address that lies in the same subnet as the MQTT broker.
3. If you are using a cloud service like AWS, parameterize a router and a DNS server.
4. Connect the CPU and the MQTT broker via Ethernet.

Note For the secured MQTT communication via TLS you need a S7-1500 CPU from firmware version 2.0 or a S7-1200 CPU from firmware version V4.3.

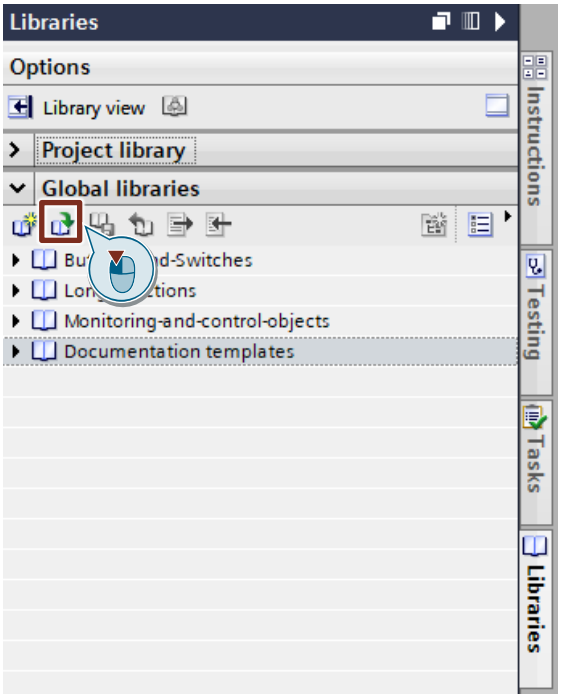
2.3 Integration of the function block in the user program

The block "LMQTT_Client" and the required data types are available in the library "LMQTT_Client".

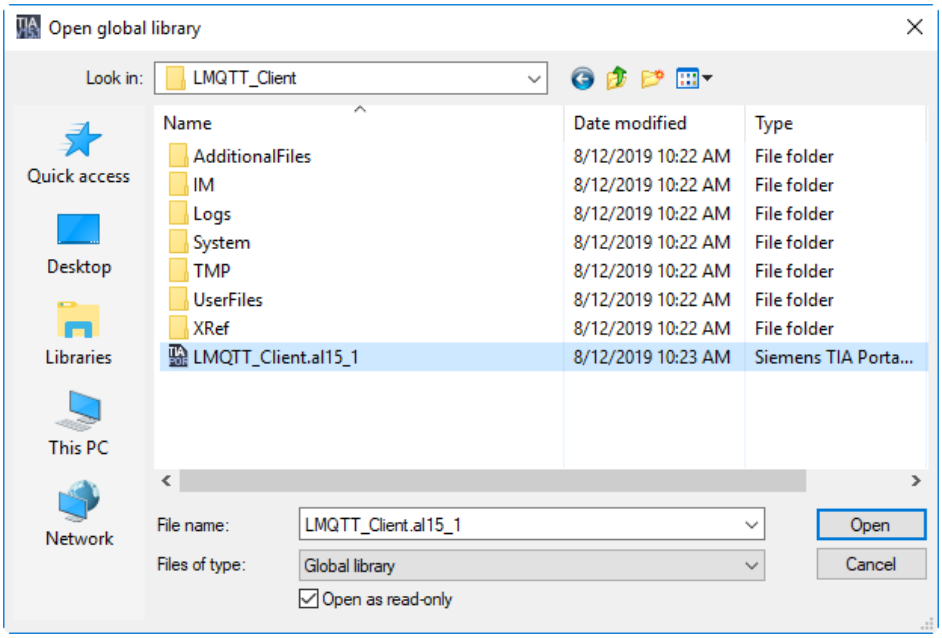
2.3.1 Opening the global library "LMQTT_Client"

Note For this section you have to download the library "LMQTT_Client.zip" and extract it into a directory of your choice.

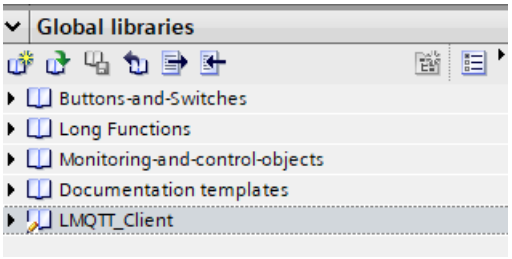
1. In the TIA Portal project, click the "Libraries" task card and open the "Global Libraries" palette.
2. Click on the "Open global library" button.
   The "Open global library" dialog is opened.



3. Select the global library "LMQTT_Client" and confirm the selection via the button "Open".

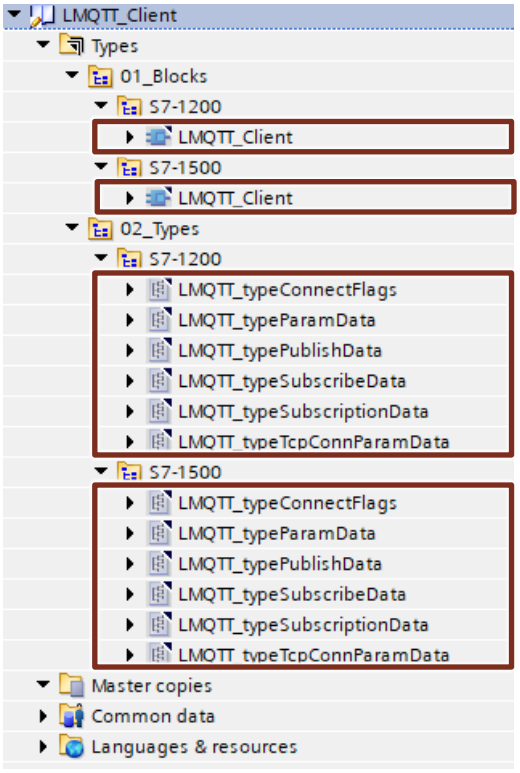


4. The library "LMQTT_Client" is opened and displayed under the palette "Global libraries".

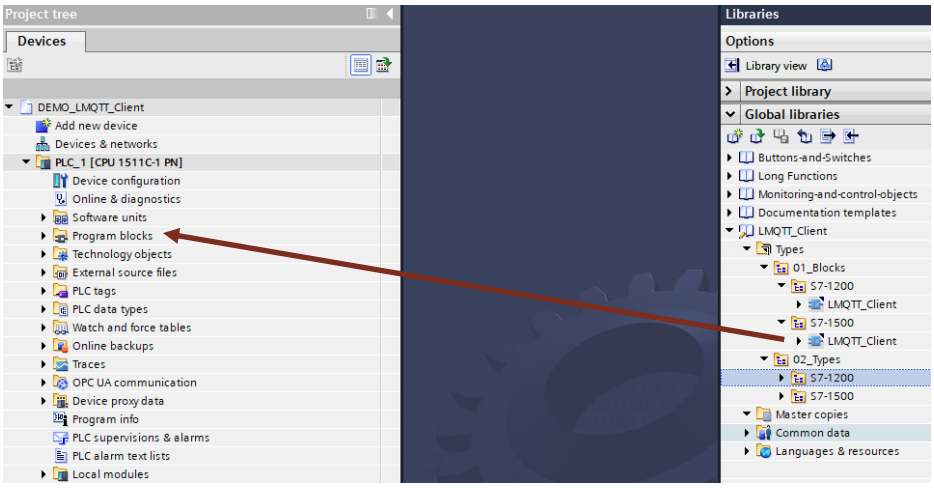


2.3.2 Copying function blocks and data types to the user program

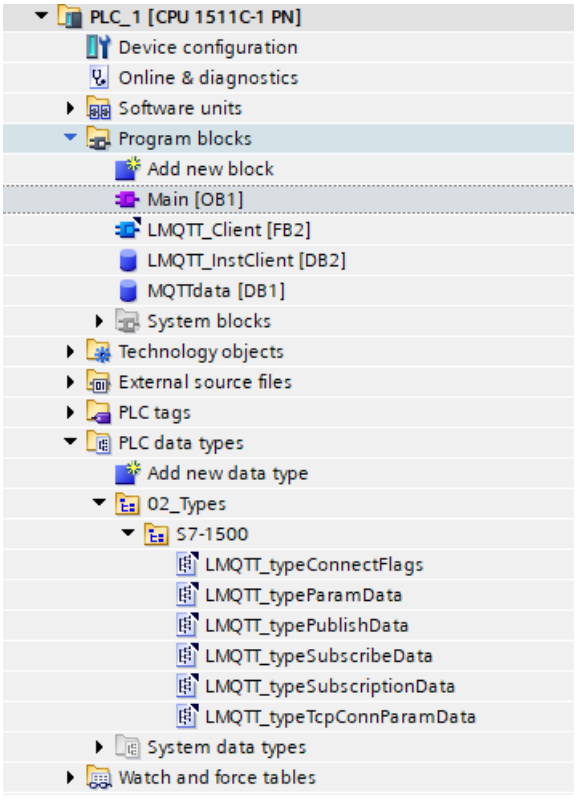
1. In the library "LMQTT_Client" you can find the FB "LMQTT_Client" for the S7-1200 CPUs and S7-1500 CPUs under "Types > 01_Blocks".
   In the library "LMQTT_Client" you can find the PLC data types using by the FB "LMQTT_Client" under "Types > 02_Types".



2. Insert the function block for your CPU via drag & drop into the folder "Program blocks" of your device, e.g. S7-1500 CPU.



3. The data types using by the FB "LMQTT_Client" are insert into the folder "PLC data types" of your device, e.g. S7-1500 CPU, automatically.

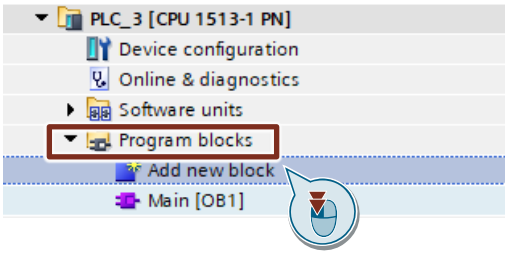


2.3.3 Creating global data block

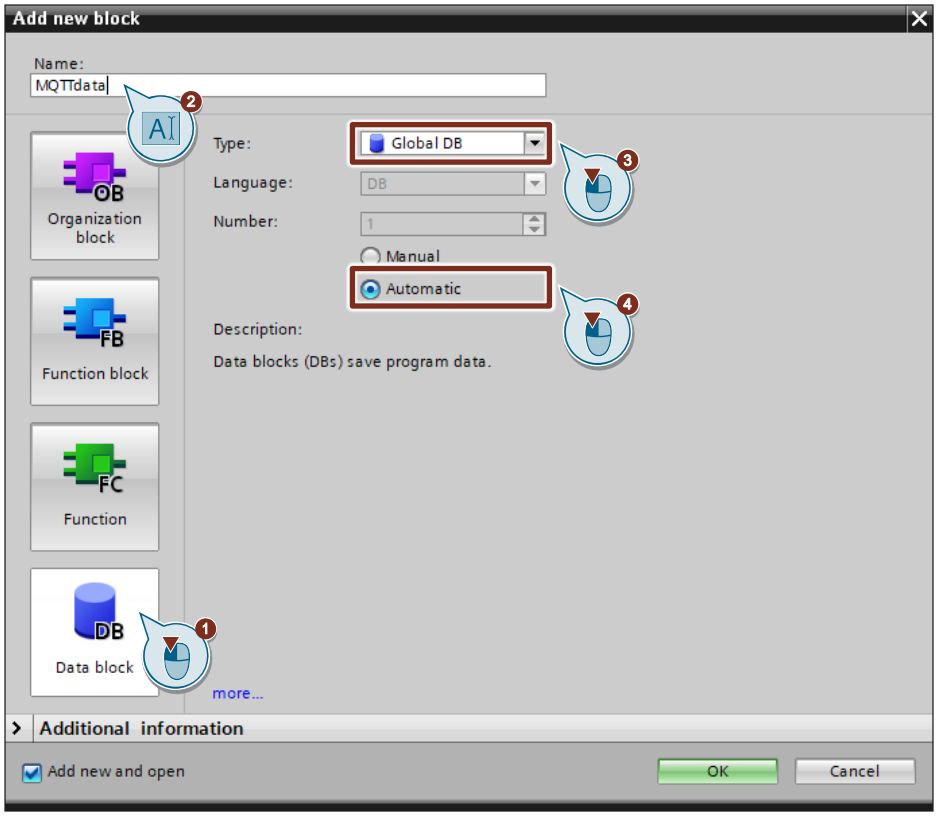
This section shows you how to create a global data block (DB). This DB is used to store the following data:

- TCP connection parameters
- MQTT connection parameters
- Topic and message to be sent to the MQTT broker (publish)
- Received data, i.e. message and name of the subscribed topic (subscribe)

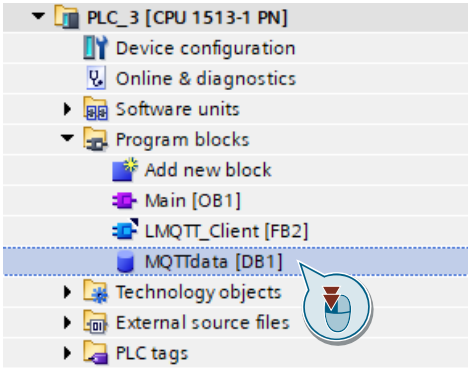
1. Navigate in the "Project tree" to the device folder of the CPU.
2. Open the "Program blocks" folder and double-click the "Add new block" command.
   The dialog "Add new block" opens.



3. Make the following settings and then confirm your entries with the "OK" button.
   - Select the symbol "Data block".
   - Select "Global DB" as the type.
   - Enter the name of the DB.
   - Activate the "Automatic" radio button for automatic number assignment. The number of the global DB is assigned by the TIA Portal.



4. Double-click the newly inserted global data block to open it.



5. Double-click "<Add new>" to create the tags accordingly.

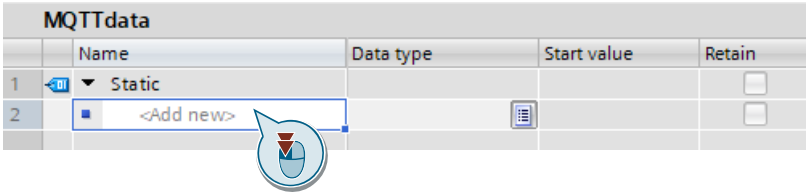
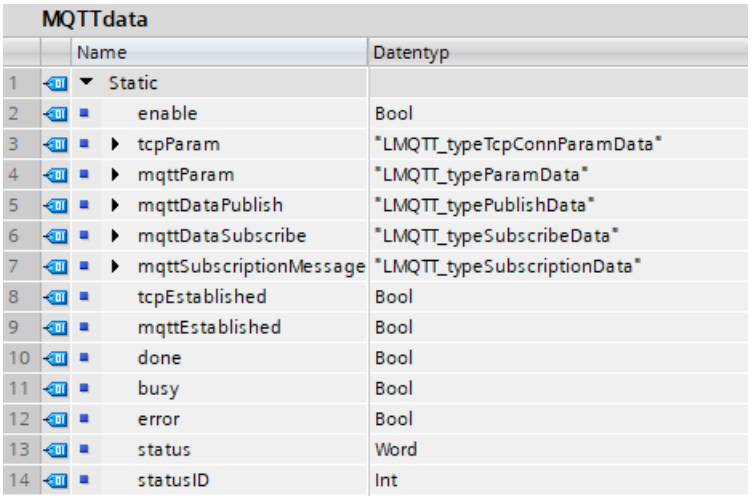


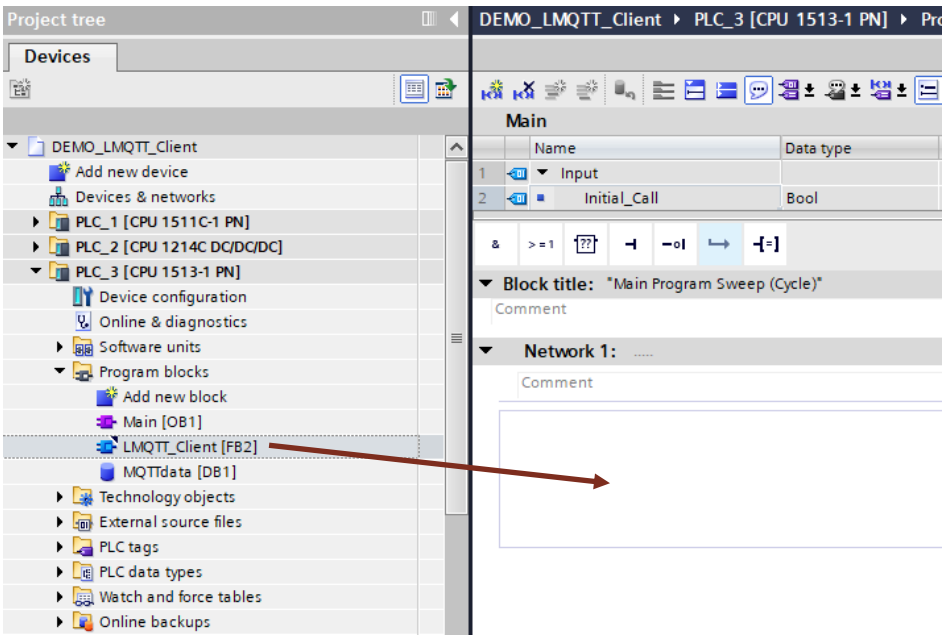
Table 2-10

Name	Data type	Interconnection at the FB
enable	Bool	Input "enable"
tcpParam	LMQTT_typeTCPConnParamData	Input "tcpConnParam"
mqttParam	LMQTT_typeParamData	Input "mqttParam""
mqttDataPublish	LMQTT_typePublishData	Input "publishData"
mqttDataSubscribe	LMQTT_typeSubscribeData	Input "subscribeToTopic"
mqttSubscriptionMessage	LMQTT_typeSubscriptionData	Output "subscription Message"
tcpEstablished	Bool	Output "tcpEstablished"
mqttEstablished	Bool	Output "mqttEstablished"
done	Bool	Output "done"
busy	Bool	Output "busy"
Error	Bool	Output "error"
status	Word	Output "status"
statusID	Int	Output "statusID"

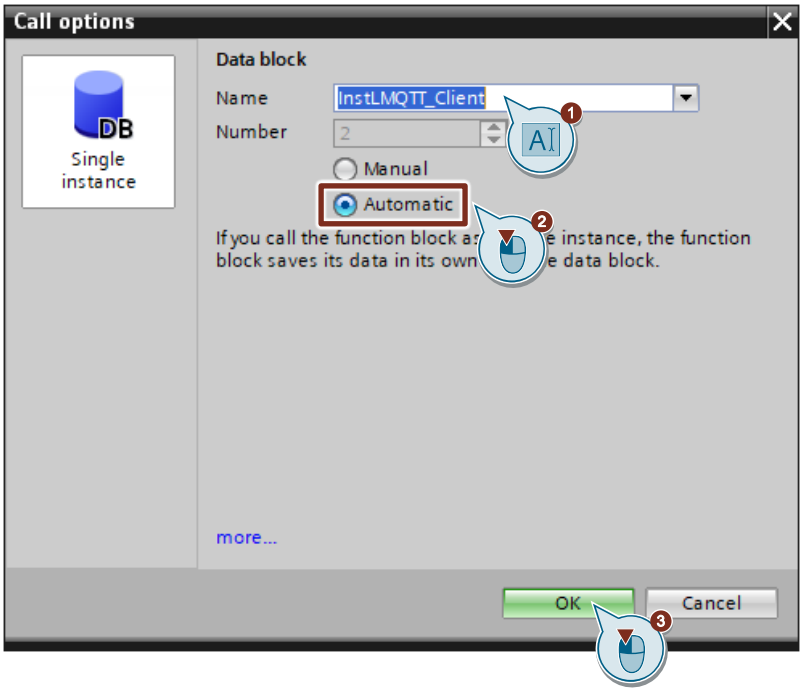


2.3.4 Calling function blocks in the user program

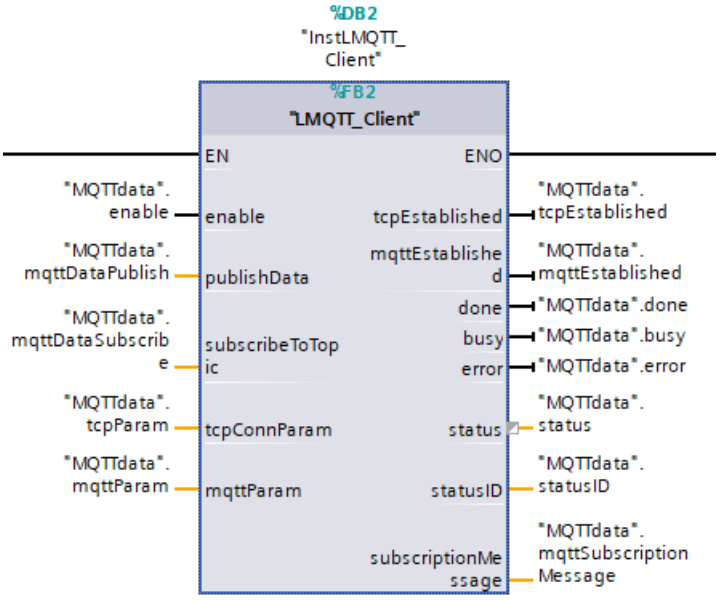
1. In the "Project tree" open the folder "Program blocks" of your CPU
2. Double-click on the block "Main [OB1]" to open the corresponding program editor.
3. Drag & drop the FB "LMQTT_Client" from the project tree to any OB1 network.



4. The dialog "Call options" for generating the instance DB of the FB "LMQTT_Client" opens automatically.
5. Make the following settings and then confirm your entries with the "OK" button.
   - Enter the name of the instance DB.
   - Activate the "Automatic" radio button for automatic number assignment. The number of the instance DB is assigned by the TIA Portal.
   - Accept the settings with "OK".



6. Assign the tags that you have created in the global data block to the inputs and outputs of the FB (see section 2.3.3).



2.4 Configuration of the security feature

Note You only need to configure the security feature if you are using a secure MQTT connection via TLS.

Note In this application example, the MQTT broker does not authenticate the MQTT client. Only the CA certificate of the MQTT broker is required to authenticate the MQTT broker.
If you have configured the MQTT broker to require MQTT client authentication, you must also import the client certificate.
The client certificate must be signed by the same CA as the server certificate.

Encryption via SSL/TLS works via certificates. A certificate is a public key signed by its owner that guarantees its authenticity and integrity. To authenticate the broker, the MQTT client requires the CA certificate of the broker.

This section shows you how to import the certificate of the MQTT broker into the CPU (MQTT client). Encrypted MQTT communication is only possible with this certificate.

Prerequisite for TLS/SSL encryption

To set up a secure MQTT communication between the SIMATIC S7 CPU (MQTT client) and an MQTT broker in your network, the following points must be fulfilled:

- The MQTT-Broker is installed and preconfigured for the TLS procedure.
- The necessary CA certificate of the MQTT broker is available to you.
- The time of the CPU is set to the current time.
  A certificate always contains a period of time in which it is valid. To be able to encrypt with the certificate, the time of the S7 CPU must also be within this period. With a brand new S7-CPU or after an overall reset of the S7-CPU, the internal clock is set to a default value that lies outside the certificate runtime. The certificate is then marked as invalid.

2.4.1 Using the global certificate manager in the TIA Portal

You must import the CA certificate of the MQTT broker into STEP 7 (TIA Portal).

In the TIA Portal, the certificates are managed in the global certificate manager. The certificate manager contains an overview of all certificates used in the project. In the certificate manager, for example, you can import new certificates and export, renew, or replace existing certificates. Each certificate is assigned an ID that can be used to reference the certificate in the program modules.

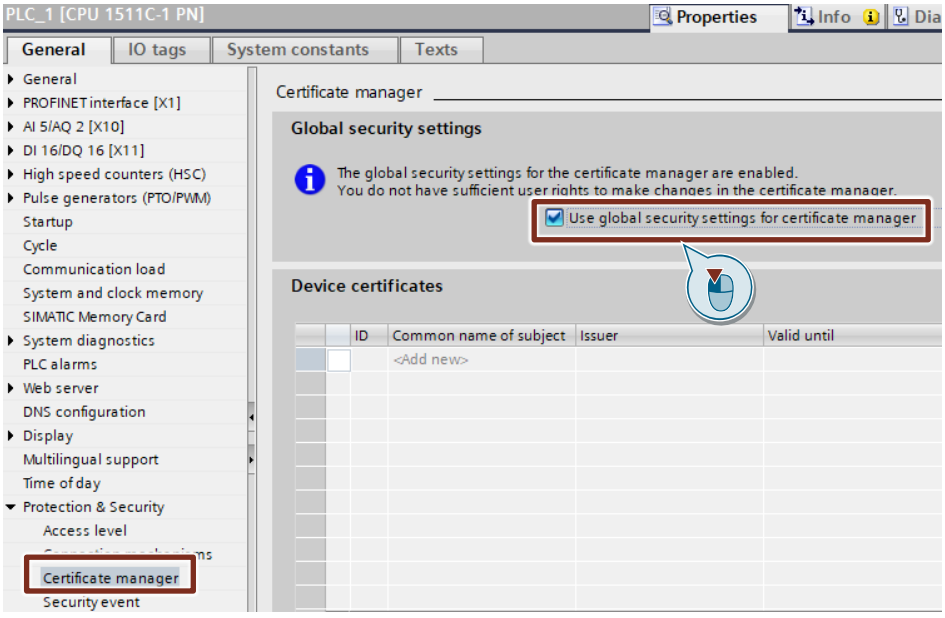
Activating the global certificate manager

If you do not use the certificate manager in the security settings, you only have access to the local certificate store of the CPU. You then have no access to imported certificates from external devices.

To import and use the CA certificate of the MQTT broker, you must activate the global certificate manager.

1. In the device or network view select the CPU. The properties of the CPU are displayed in the inspection window.

2. In the area navigation of the "Properties" tab, select "Protection & Security > Certificate Manager". Activate the function "Use global security settings for certificate manager".



Result

The new entry "Security Settings" appears in the project navigation.

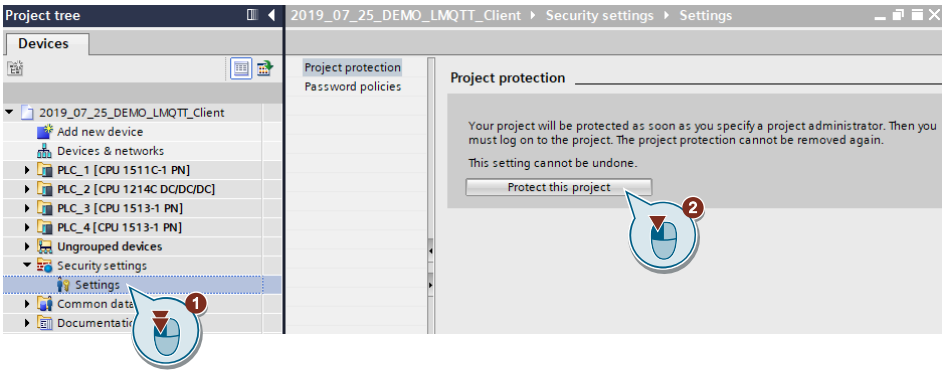
Logging on users

After you have enabled the global security settings for the certificate manager, you must log in to the security settings. You cannot access the global certificate manager without logging in.

Log on as a security user for the security settings as described below:

1. Double-click on the entry "Settings" in the project navigation under "Security settings".

2. The user management editor opens, and the project protection area is displayed.
   Click the "Protect this project" button.



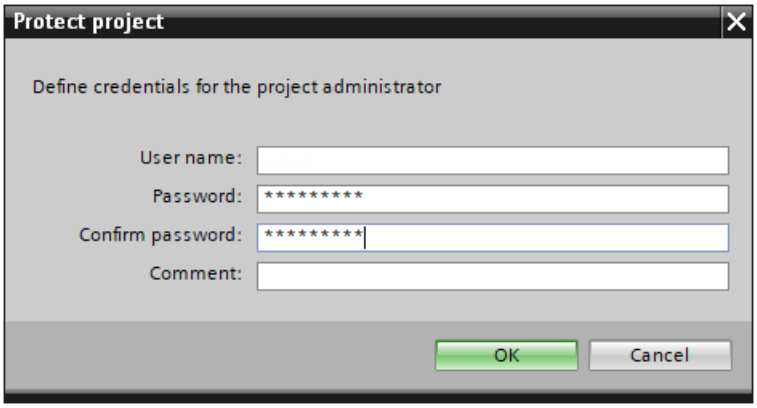
3. This opens the dialog "Protect Project".
   Enter a username and password.

   The password must comply with the following guidelines:

   - Password length: A minimum of 8 characters, a maximum of 128 characters
   - At least one upper-case letter
   - At least one special character (special characters § and ß are not allowed)
   - At least one number

   Enter the password again to confirm.

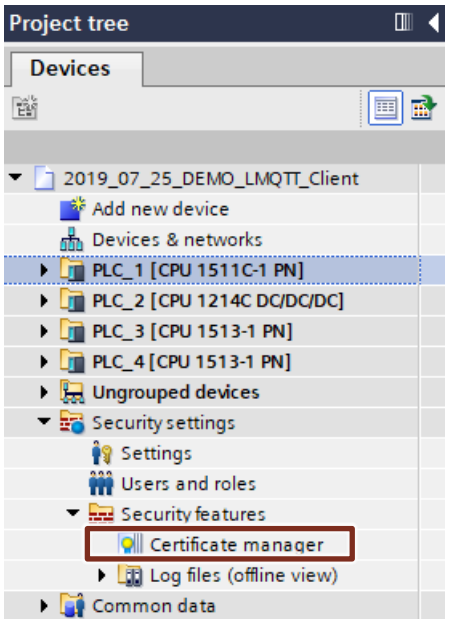
4. You may enter a comment if required.
   Confirm your entries with "OK".



Result

You have activated the user management. You are logged in as a project administrator and can use the security settings.
If you have logged in, a line "Certificate manager" appears under the entry "Security settings > Security features".

Figure 2-1



Using the global certificate manager

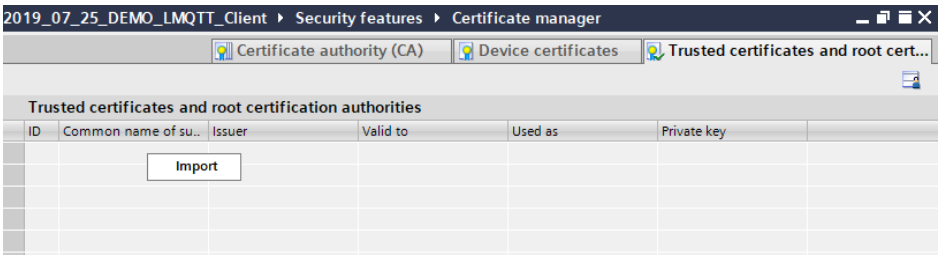
With the global certificate manager, you now have the option of importing third-party certificates into the TIA portal. By double-clicking on the line "Certificate manager" you gain access to all certificates in the project, divided into the following tabs:

- "Certification authority (CA)"
- "Device certificates"
- "Trusted certificates and root certification authorities"

1. Double-click on the "Certificate manager" entry in the project navigation under "Security settings > Security features".

2. Select the appropriate registry for the certificate you want to import, for example, "Trusted certificates and root certification authorities".

3. To open the context menu, right-click in the tab. Click "Import".

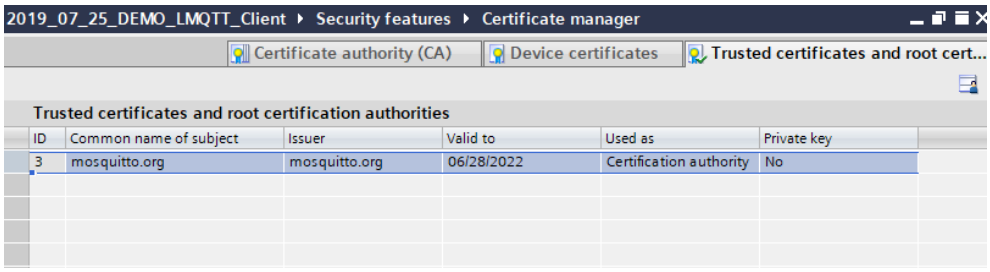


4. Select the export format of the certificate:
   - CER, DER, CRT or PEM for certificates without a private key
   - P12 (PKCS12 archive) for certificates with a private key.

   Click on "Open" to import the certificate.

Result
The CA certificate of the MQTT broker is now located in the global certificate manager.

Figure 2-2



Note

If the MQTT broker also requires authentication of the MQTT client, you must import the client certificate.
Observe the following information:

- The client certificate must be signed by the same CA as the server certificate.
- The client certificate must be imported as a PK12 container (with certificate and private key) into the global certificate manager.
- The client certificate must be imported into the "Device certificates" table.

2.4.2 Using the local certificate manager of the CPU

The CA certificate is currently only located in the global certificate manager of the TIA Portal. Certificates imported via the certificate manager into the global security settings are not automatically assigned to the corresponding modules.

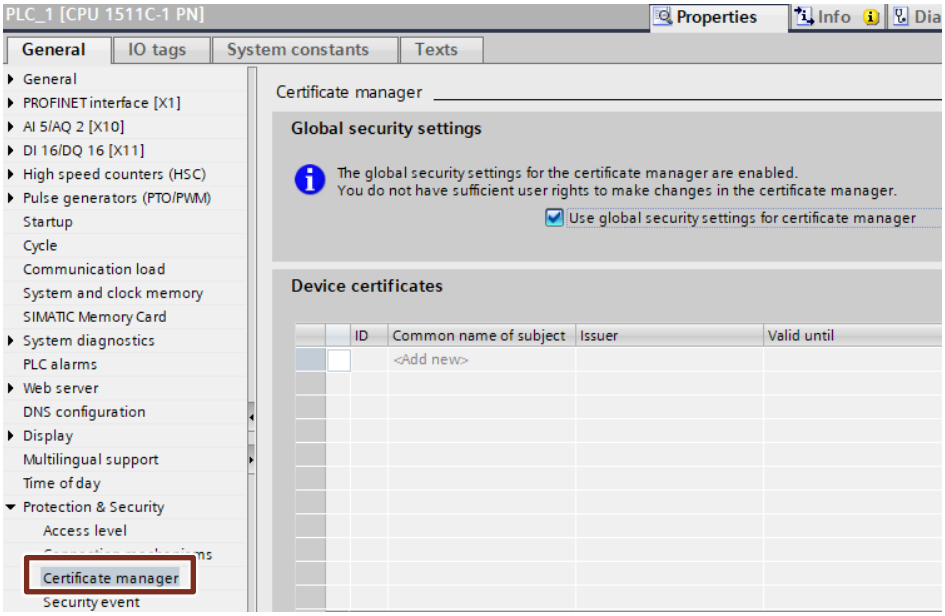
To authenticate the MQTT broker, you have to load the CA certificate into the CPU. Only those device certificates that you have assigned to the module as device certificates via the local certificate manager are loaded onto the module.

This assignment is made in the local security settings of the module in the entry "Certificate manager" via the table editor "Device certificates". The certificates of the global certificate manager are available for the certificate assignment.

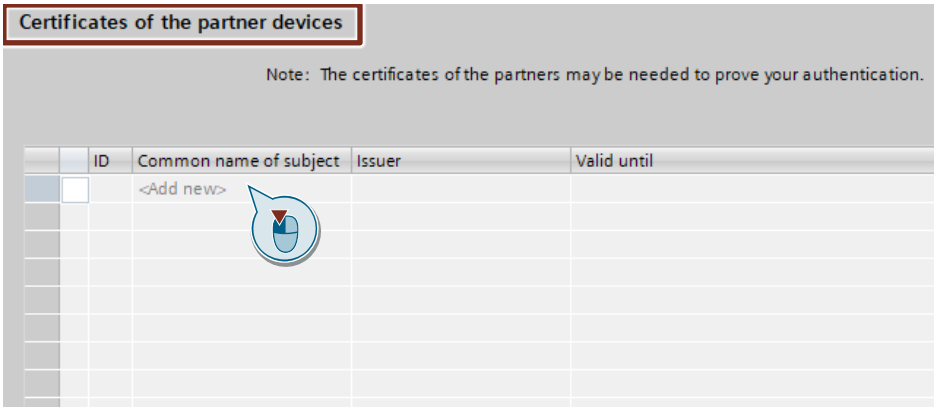
The following steps show you how to assign the CA certificate from the global certificate manager to the CPU.

1. In the device or network view select your CPU. The properties of the CPU are displayed in the inspection window.

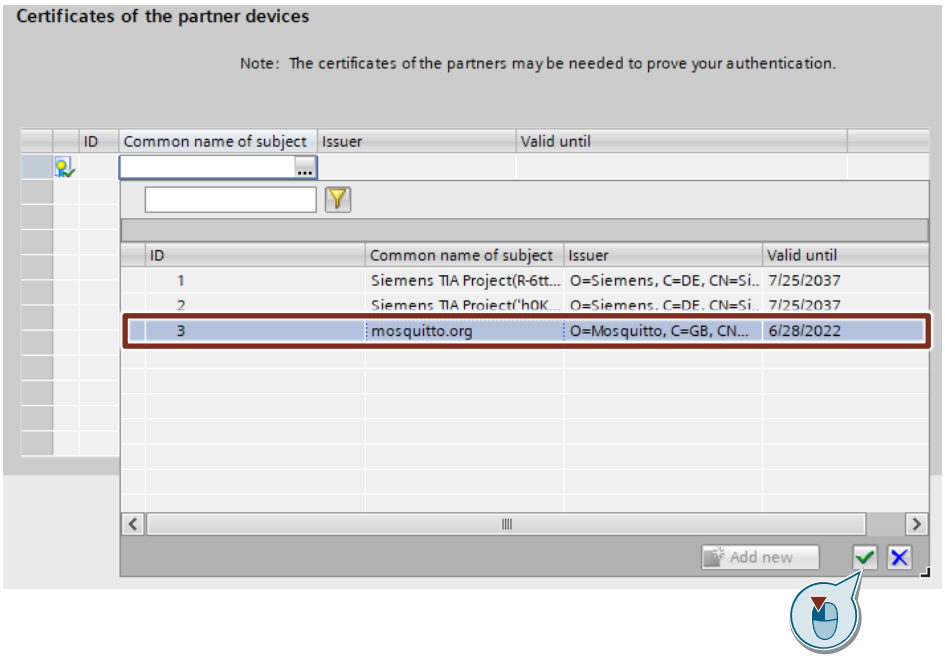
2. To add the CA certificates, select the entry "Certificate manager" in the area navigation of the "Properties" tab under "Protection & Security".



3. Under "Certificates of the partner devices". Click "Add" in the table of certificates. This inserts a new row into the table.



4. Click in the new row. The selection for new certificates opens. Select the previously imported CA certificate from the global certificate manager and click the green check mark.



Result

The selected certificate was assigned to the CPU and provided with an ID. The ID is the number of the certificate. You enter this value in the connection parameters for the parameter "idTlsServerCertificate" (see Table 2-3).

Figure 2-3

Note If the MQTT broker additionally requires authentication of the MQTT client, you must also assign the imported client certificate to the CPU (section "Device certificates"). You enter the value of the ID in the connection parameters for the parameter "idTlsOwnCertificate" (see Table 2-3).

2.5 Parameter assignment and operation

Setting the parameters

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

All parameters that you can define yourself are in the "MQTTdata" global 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 the domain name of the MQTT broker. The domain name must end with a ".".
- remote port on which the MQTT broker receives the messages
  - unsecured connection: remote port 1883
  - secured connection: remote port 8883
- Local port in the CPU, if assigned “0”, are free port will be determined by the system
- Parameters for secure communication
  - status of the security feature (On/Off) for this connection.
  - ID of the CA certificate (only relevant for a secure connection)
  - ID of your own certificate, if the MQTT broker also authenticates the client (only relevant for a secure connection).
- 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 the TCP connection is to be established via the fully qualified domain name, you must configure a DNS server in the CPU.

Operating the application example

Once you have set all parameters and added the CA certificate of the MQTT broker to the local certificate manager of the CPU, 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. To do this, trigger the variable "enable" on the function block "LMQTT_Client". As long as the Input is “True”, the connection stays established. If you reset the input Enable to “False”, the communication will be disconnected.

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 or receive an MQTT message on a subscribed topic. To do this, trigger the input variable "publishData" or "subscribeToTopic".

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 Section 2.6.

2.6 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 and have the following meaning.

Table 2-11

Value	Description
-1	TCP_DISCONNECT
2	TCP_CONNECTING
3	TCP_CONNECTED
9	MQTT_DISCONNECT_STATE
10	MQTT_CONNECT_FLAG_CHECK_STATE
11	MQTT_CONNECT_STATE
12	MQTT_CONNACK_STATE
13	MQTT_PUBLISH_STATE
14	MQTT_PUBACK_STATE
15	MQTT_SUBSCRIBE_STATE
17	MQTT_PING
19	MQTT_PUBREC_ACK_STATE
20	MQTT_PUBCOMP_STATE
21	MQTT_SUBACK_STATE
22	MQTT_UNSUBSCRIBE_STATE
23	MQTT_UNSUBACK_STATE
26	MQTT_RECEIVED_QOS1_STATE
54	MQTT_MSG_TYPE_PUB_INC_MESSAGE_BAD
-13	MQTT_TIMEOUT_ID
-14	MQTT_BAD_ID
-16	TIME_MONITORING_ID

"status"

The output parameter "status" displays the error code:

Table 2-12

statusID	status	Description	Remedy
-1	Status message from the "TDISCON" block	See manual	−
2	Status message from the "TCON" block	See manual	Check availability of the MQTT broker: • IP address • Port • Firewall
3	Status message from the "TRCV" block	See manual	Check network connection
9	Status message from the "TSEND" block	See manual	−
10	W#16#80F0	Error with the flags "Will" in the data type "LMQTT_typeConnectFlags": The parameter "willRetain" was set to "TRUE" without the parameter "will" being set to "TRUE".	Check the following parameters in the data type "LMQTT_typeConnectFlags": • will • willRetain
	W#16#80F1	Error with the flags "WillQoS" in the data type "LMQTT_typeConnectFlags": • The parameter "willQoS1" or "willQoS2" was set to "TRUE" without the parameter "will" being set to "TRUE". • The parameters "willQoS1" and "willQoS2" were simultaneously set to "TRUE".	Check the following parameters in the data type "LMQTT_typeConnectFlags": • will • willQoS1 • willQoS2
	W#16#80F3	Error during the flag "keepAlive"	KeepAlive must exceed 2 s.
11	Status message from the "TSEND" block	See manual	-

statusID	status	Description	Remedy
12	1	• The MQTT broker does not support the level of the MQTT protocol requested by the MQTT client. • Clean session does not match.	Check access data in data type "LMQTT_typeParamData"
	2	The MQTT broker does not allow the client ID.	
	3	MQTT service not available.	
	4	The data in the username and password are incorrect.	
	5	The MQTT client is not authorized to connect.	
13	W#80F4	The server quality QoS of the message to be sent is not correct.	The following values are possible for QoS: • QoS = 0: fire and forget • QoS = 1: Message arrives at least once • QoS = 2: Message arrives exactly once
	W#80F5	The name of the topic to which you want to send messages has not been defined.	In the "publishTopic" parameter of the "LMQTT_typePublishData" data type, enter the name of the topic to which messages are to be sent.
14	W#80F2	Wrong packet ID in "PUBACK" packet (Publish Acknowledgement)	−
15	W#80F4	The server quality QoS of the message received is not correct.	The following values are possible for QoS: • QoS = 0: fire and forget • QoS = 1: Message arrives at least once • QoS = 2: Message arrives exactly once
	W#80F5	The name of the topic to be subscribed to has not been defined.	In the "subscriptionTopic" parameter of the "LMQTT_typeSubcripeData" data type, enter the name of the topic to be subscribed to.
17	Status message from the "TSEND" block	See manual	−
19	W#80F2	Wrong packet ID in "PUBREC" packet (Publish Received)	−
20	W#80F2	Wrong packet ID in "PUBCOMP" packet (Publish Complete)	−

statusID	status	Description	Remedy
21	W#80F7	The SUBACK packet (Subscribe Acknowledgement) of the MQTT broker is faulty.	−
	W#80F2	Wrong packet ID in "SUBACK" packet (Subscribe Acknowledgement)	−
22	Status message from the "TSEND" block	See manual	−
23	W#80F2	Wrong packet ID in "UNSUBACK" packet (Unsubscribe Acknowledgement)	−
26	Status message from the "TSEND" block	See manual	−
-13	W#80F3	Error during the flag "keepAlive" No response from the server after the KeepAlive was triggered.	−
-14	Value remaining length	Remaining length in one of the following packets invalid: • PUBREC: Publish Received • PUBACK: Publish Acknowledgement • CONNACK: Connect Acknowledgement • SUBACK: Subscribe Acknowledgement • UNSUBACK: Unsubscribe Acknowledgement • PUBCOMP: Publish Completed	−
-16	Watchdog time	Timeout	Check connection parameters
54	W#80F6	A message is received which simultaneously has server quality QOS = 1 and QOS = 2.	

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).

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 MQTT-Broker (Publish mechanism)
- Subscribe messages which follow a certain topic (Subscriber mechanism) at the MQTT broker
- Unsubscribe yourself from subscribed messages
- Disconnect from the broker

Note The function block "LMQTT_Client" in this application example supports the following functions:

- Unsubscribe from the MQTT broker.
- Publish mechanism
- Subscribe and unsubscribe mechanismus
- Ping mechanism
- Unsubscribe from the MQTT broker.

MQTT broker

An MQTT broker is the central component of MQTT and can be a program or a device. The MQTT 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 MQTT broker has the following functions:

- Accept network connections from the MQTT 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 service qualities 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 MQTT broker acknowledges receipt of the message.
- QoS "2": In the highest level 2, the MQTT broker guarantees by multiple handshake with the MQTT 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 MQTT clients if the connection to a MQTT client has been disconnected accidentally.

Each MQTT client can specify its last will while connecting to the MQTT broker and notify the MQTT broker. This last will is built like a normal MQTT message, including topic, QoS and payload. The MQTT broker saves the last will. As soon as the MQTT broker notices that the connection with the MQTT client in question has been abruptly terminated, the MQTT 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 MQTT client has been disconnected.

KeepAlive

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

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

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

When a message or a ping packet has been sent to the MQTT 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 an MQTT client is interrupted, the broker can cache new messages for this client for later delivery.

Retained messages

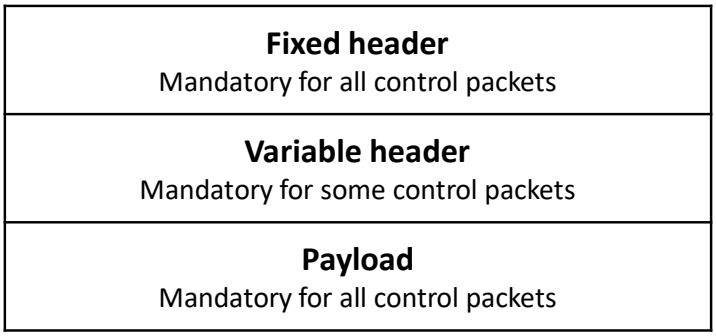
The first time an MQTT client subscribes to a topic, it usually gets a message only when another MQTT client sends a message with the subscribed topic the next time. With "Retained messages", the subscriber receives the last value sent to the topic prior to its subscription request, delivered immediately.

3.1.4 Structure of the MQTT control packets

Most MQTT control packets 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



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 packets. 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).

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

3.1.5 MQTT connection

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

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

The MQTT 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

MQTT control packet "CONNECT"

Table 3-1 shows the structure of the "fixed header" of the "CONNECT" packet.

Table 3-1

Fixed header								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Byte 1	Identifier number for MQTT control packet type = 1 (dec)				Reserve			
Byte 2	Remaining length: The number of following bytes after the "fixed header" = "variable header" + "payload"							

A "CONNECT" packet contains the following areas in the "variable header":

1. Report name: The report name "MQTT" is transmitted as UTF-8 string.
2. Report level: 4 (dec)
3. Connect flags: The "Connect Flags" byte contains a number of parameters that specify the behavior of the MQTT connection. In addition, the "Connect Flags" byte also shows which optional fields are present in the "payload" or not. The connection type can be regulated with the "Clean Session" flag.
4. Keep alive: The KeepAlive time determines the time interval in which the MQTT client is obligated to report to the MQTT 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 MQTT broker disconnects from the client.

Table 3-2 shows the structure of the "variable header" of the "CONNECT" packet.

Table 3-2

Variable header								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Report name								
Byte 1	MSB length = 0 (dec)							
Byte 2	LSB length = 4 (dec)							
Byte 3	'M'							
	0	1	0	0	1	1	0	1
Byte 4	'Q'							
	0	1	0	1	0	0	0	1
Byte 5	'T'							
	0	1	0	1	0	1	0	0
Byte 6	'T'							
	0	1	0	1	0	1	0	0
Report level								
Byte 7	Report level = 4 (dec)							
Connect flags								
Byte 8	Username flag	Password flag	Will retain flag	Will QoS flag		Will flag	Clean session flag	Reserve
Keep alive								
Byte 9	Keep alive MSB							
Byte 10	Keep alive LSB							

In "Payload" the existing fields appear in the following order:

- Client ID: The client ID is used to identify the client at the MQTT broker The client ID must appear as the first field in the "Payload".
- Will topic: The field appears optionally if the "Will" flag is set to "TRUE".
- Will message: The field appears optionally if the "Will" flag is set to "TRUE".
- Username: The field appears optionally if the "Username" flag is set to "TRUE".
- Password: The field appears optionally if the "Password" flag is set to "TRUE".

MQTT control packet "CONNACK"

Table 3-3 shows the structure of the "fixed header" of the "CONNACK" packet.

Table 3-3

Fixed header								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Byte 1	Identifier number for MQTT control packet type = 2 (dec)				Reserve			
Byte 2	Remaining length: The number of following bytes after the "fixed header" = 2 bytes							

Table 3-4 shows the structure of the "variable header" of the "CONNACK" packet.

Table 3-4

Variable header								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Connect acknowledge flags								
Byte 1	Reserve							Session Present
Connect Return Code								
Byte 2	• 0x00 = The MQTT broker accepts the connection. The MQTT broker does not support the level of the MQTT protocol requested by the client. • 0x01 = The MQTT broker does not support the level of the MQTT protocol requested by the MQTT client. • 0x02: The MQTT broker does not allow the client ID. • 0x03 The MQTT service is not available. • 0x04: The data in the username and password are incorrect. • 0x05: The MQTT client is not authorized to connect.							

3.1.6 MQTT-push mechanism

Once an MQTT client connects to the MQTT broker, it can send messages to the MQTT 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".

"PUBLISH" packet

Table 3-5 shows the structure of the "fixed header" of the "PUBLISH" packet.

Table 3-5

Fixed header								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Byte 1	Identifier number for MQTT control packet type = 3 (dec)				DUP flag	QoS level		Retain flag
	0	0	1	1	X	X	X	X
Byte 2	Remaining length: The number of following bytes after the "fixed header" = "variable header" + payload							

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

- QoS = 0 (dec): The message will be sent only once. The send job ends here.
- QoS = 1 (dec): The message will be sent at least once. The MQTT broker acknowledges the "PUBLISH" packet with a "PUBACK" packet.
- QoS = 2 (dec): The message will be sent exactly once. The MQTT broker acknowledges the "PUBLISH" packet with a "PUBREC" packet. This is followed by another handshake between MQTT client and MQTT broker. The client answers the "PUBREC" packet with a "PUBREL" packet. The MQTT broker completes the double handshake with a "PUBCOM" packet.

Note You can find further information on Quality Assurance QoS in Section 3.1.3.

The "variable header" of the "Publish" packet contains the following fields:

- Name of the topic
- Packet ID

The "Payload" contains the message text.

"PUBACK" packet (Publish Acknowledgement)

The MQTT broker responds to the "PUBLISH" packet with QoS=1 with the "PUBACK" packet.

Table 3-6 shows the structure of the "fixed header" of the "PUBACK" packet.

Table 3-6

Fixed header								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Byte 1	Identifier number for MQTT control packet type = 4 (dec)				Reserve			
	0	1	0	0	0	0	0	0
Byte 2	Remaining length: The number of following bytes after the "fixed header" = 2 bytes							

The "variable header" of the "PUBACK" packet contains the packet ID.

The "PUBACK" packet has no "payload".

"PUBREC" packet (Publish Received)

The MQTT broker responds to the "PUBLISH" packet with QoS=2 with the "PUBREC" packet.

Table 3-7 shows the structure of the "fixed header" of the "PUBREC" packet.

Table 3-7

Fixed header								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Byte 1	Identifier number for MQTT control packet type = 5 (dec)				Reserve			
	0	1	0	1	0	0	0	0
Byte 2	Remaining length: The number of following bytes after the "fixed header" = 2 bytes							

The "variable header" of the "PUBREC" packet contains the packet ID.

The "PUBREC" packet has no "payload".

"PUBREL" packet (Publish Release)

The MQTT client responds to the "PUBREC" packet with the "PUBREL" packet.

Table 3-8 shows the structure of the "fixed header" of the "PUBREL" packet.

Table 3-8

Fixed header								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Byte 1	Identifier number for MQTT control packet type = 6 (dec)				Reserve			
	0	1	1	0	0	0	1	0
Byte 2	Remaining length: The number of following bytes after the "fixed header" = 2 bytes							

Note

The reserve bits in the "fixed header" must be set as follows:

- Bit 3 = 0
- Bit 2 = 0
- Bit 1 = 1
- Bit 0 = 0

The "variable header" of the "PUBREL" packet contains the packet ID.

The "PUBREL" packet has no "payload".

"PUBCOMP" packet (Publish Complete)

The MQTT broker responds to the "PUBREL" packet with the "PUBCOMP" packet.

Table 3-9 shows the structure of the "fixed header" of the "PUBCOMP" packet.

Table 3-9

Fixed header								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Byte 1	Identifier number for MQTT control packet type = 7 (dec)				Reserve			
	0	1	1	1	0	0	0	0
Byte 2	Remaining length: The number of following bytes after the "fixed header" = 2 bytes							

The "variable header" of the "PUBCOMP" packet contains the packet ID.

The "PUBCOMP" packet has no "payload".

3.1.7 MQTT sub-mechanism

Once an MQTT client has connected to the MQTT broker, it can create or unsubscribe from subscriptions.

"SUBSCRIBE" packet

To create a subscription, the MQTT client uses the "SUBSCRIBE" packet. A list of the topics that the MQTT client would like to subscribe to is stored in the "Payload".

Table 3-10 shows the structure of the "fixed header" of the "SUBSCRIBE" packet.

Table 3-10

Fixed header								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Byte 1	Identifier number for MQTT control packet type = 8 (dec)				Reserve			
	1	0	0	0	0	0	1	0
Byte 2	Remaining length: The number of following bytes after the "fixed header" = "variable header" + "payload"							

Note

The reserve bits in the "fixed header" must be set as follows:

- Bit 3 = 0
- Bit 2 = 0
- Bit 1 = 1
- Bit 0 = 0

The "variable header" of the "SUBSCRIBE" packet contains the packet ID.

Table 3-11 shows the structure of the "payload" of the "SUBSCRIBE" packet.

Table 3-11

Payload								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Topic name								
Byte 1	MSB length							
Byte 2	LSB length							
Byte 3…n	Topic name							
Requested service quality QoS								
Byte n+1	Reserve							QoS level Possible values: • 0 (dec) • 1 (dec) • 2 (dec)

"SUBACK" packet (Subscribe Acknowledgement)

The MQTT broker responds to the "SUBSCRIBE" packet with the "SUBACK" packet.

Table 3-12 shows the structure of the "fixed header" of the "SUBACK" packet.

Table 3-12

Fixed header								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Byte 1	Identifier number for MQTT control packet type = 9 (dec)				Reserve			
	1	0	0	1	0	0	0	0
Byte 2	Remaining length: The number of following bytes after the "fixed header" = "variable header" + "payload"							

The "variable header" of the "SUBACK" packet contains the packet ID.

Table 3-13 shows the structure of the "payload" of the "SUBACK" packet.

Table 3-13

Payload								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Return code								
Byte 1	• 0x00 Successful: Maximum service quality QoS 0 • 0x01: Successful: Maximum service quality QoS 1 • 0x02: Successful: Maximum service quality QoS 2 • 0x80: Error							

"UNSUBSCRIBE" packet

To unsubscribe from a subscription, the MQTT client uses the "UNSUBSCRIBE" packet. A list of the topics that the MQTT client would like to unsubscribe from is stored in the "Payload".

Table 3-14 shows the structure of the "fixed header" of the "UNSUBSCRIBE" packet.

Table 3-14

Fixed header								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Byte 1	Identifier number for MQTT control packet type = 10 (dec)				Reserve			
	1	0	1	0	0	0	1	0
Byte 2	Remaining length: The number of following bytes after the "fixed header" = "variable header" + "payload"							

Note

The reserve bits in the "fixed header" must be set as follows:

- Bit 3 = 0
- Bit 2 = 0
- Bit 1 = 1
- Bit 0 = 0

The "variable header" of the "UNSUBSCRIBE" packet contains the packet ID.

Table 3-15 shows the structure of the "payload" of the "UNSUBSCRIBE" packet.

Table 3-15

Payload								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Topic name								
Byte 1	MSB length							
Byte 2	LSB length							
Byte 3…n	Topic name							

"UNSUBACK" packet

The MQTT broker responds to the "UNSUBSCRIBE" packet with the "UNSUBACK" packet.

Table 3-16 shows the structure of the "fixed header" of the "UNSUBACK" packet.

Table 3-16

Fixed header								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Byte 1	Identifier number for MQTT control packet type = 11 (dec)				Reserve			
	1	0	1	1	0	0	0	0
Byte 2	Remaining length: The number of following bytes after the "fixed header" = "variable header" = 2 bytes							

The "variable header" of the "UNSUBACK" packet contains the packet ID.

The "UNSUBACK" packet has no "payload".

3.1.8 MQTT-ping mechanism

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

"PINGREQ" packet

Table 3-17 shows the structure of the "fixed header" of the "PINGREQ" packet.

Table 3-17

Fixed header								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Byte 1	Identifier number for MQTT control packet type = 12 (dec)				Reserve			
	1	1	0	0	0	0	0	0
Byte 2	Remaining length: The number of following bytes after the "fixed header" = 0 bytes							

The "PINGREQ" packet has no "variable header" and no "payload".

"PINGRESP" packet

The MQTT broker responds to the "PINGREQ" packet with the "PINGRESP" packet and thus signals its availability to the MQTT client.

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

Table 3-18 shows the structure of the "fixed header" of the "PINGRESP" packet.

Table 3-18

Fixed header								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Byte 1	Identifier number for MQTT control packet type = 13 (dec)				Reserve			
	1	1	0	1	0	0	0	0
Byte 2	Remaining length: The number of following bytes after the "fixed header" = 0 bytes							

The "PINGRESP" packet has no "variable header" and no "payload".

3.1.9 MQTT disconnection

An MQTT client can close the connection to an MQTT broker by sending a "DISCONNECT" packet to the MQTT broker. After the MQTT client has sent the "DISCONNECT" packet and closed the connection, it does not need to send any more MQTT control packets. When the MQTT broker receives a "DISCONNECT" packet, it deletes all "last will and testament" information. As the MQTT client is actively and voluntarily connected, the MQTT broker does not send its last wishes to the registered subscribers.

"DISCONNECT" packet

Table 3-19 shows the structure of the "fixed header" of the "DISCONNECT" packet.

Table 3-19

Fixed header								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Byte 1	Identifier number for MQTT control packet type = 14 (dec)				Reserve			
	1	1	1	0	0	0	0	0
Byte 2	Remaining length: The number of following bytes after the "fixed header" = 0 bytes							

The "DISCONNECT" packet has no "variable header" and no "payload".

3.2 How the "LMQTT_Client" 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_Client" 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 MQTT 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 MQTT broker.

Overview

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

- State machine "TCP state machine": Management of the TCP connection
- State machine "MQTT state machine": Management of the MQTT connection
- Internal state machine "MQTT job state machine": Handling of the transfer

3.2.2 State machine "TCP state machine":

The state machine "TCP state machine" 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 state machine" 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 3-20

State	Description
IDLE	In "IDLE" state, all parameters are reset. 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 parameter, 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 parameter "tcpEstablished" is set. The TCP connection persists until it is terminated with "TDISCON". If an error occurs during connection setup, the state machine switches to the "TCP_ERROR" state.
TCP_CONNECTED	In this state, the function block maintains the state until the following events occur: • The "TRCV" block detects a connection abort, e.g. by the network cable being pulled out, and reports an error. • The input parameter "enable" is reset and thus initiates the disconnection. If the "TRCV" block detects an error, the state machine changes to the "TCP_ERROR" state. The "TCP_CONNECTED" state is a prerequisite for the processing of the state machine "MQTT state machine".
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 state machine", 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: • The error message of the T-block involved is transferred at the output "status". • The number of the state in which the error occurred is output at the output "statusID". • The state machine returns to the "IDLE" state. If there is already a TCP connection, it will be disconnected in advance. The output variable "tcpEstablished" is reset. • The state machine "MQTT state machine" is set to the state "MQTT_DISCONNECTED_STATE".

Note The function block "LMQTT_Client" 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 state machine":

The state machine "MQTT state machine" is automatically started when the state machine "TCP state machine" 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 "MQTT job state machine" to send messages
- It makes sure that a PING packet is sent before the KeepAlive interval expires.

The state machine "MQTT state machine" contains the following states:

- MQTT_DISCONNECTED_STATE
- MQTT_CONNECT_FLAG_CHECK_STATE
- MQTT_CONNECT_STATE
- MQTT_CONNACK_STATE
- MQTT_CONNECTED
- MQTT_DISCONNECT
- MQTT_ERROR

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

Table 3-21

State	Description
MQTT_DISCONNECTED_STATE	As long as there is no TCP connection, the state machine is always in the "MQTT_DISCONNECTED_STATE". Only when a TCP connection has been established is the switching condition automatically activated for the "MQTT_CONNECT_FLAG_CHECK_STATE"
MQTT_CONNECT_FLAG_CHECK_STATE	In 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_STATE" without a switching condition.
MQTT_CONNECT_STATE	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 then sent to the MQTT broker with the "TSEND" block. The state machine changes to the "MQTT_CONNACK_STATE".

State	Description
MQTT_CONNACK_ STATE	If an error occurred while sending the "CONNECT" packet, the state machine will change to the "MQTT_ERROR_STATE". If the "CONNECT" packet has been successfully sent, the state is not exited until a message is received at the "TRCV" block. The MQTT client expects a "CONNACK" packet from the MQTT broker to acknowledge the "CONNECT" packet. It is checked whether the received acknowledgment is a "CONNACK" packet. When the MQTT broker has confirmed receipt of the message, the state machine changes to the state "MQTT_CONNECTED_STATE" and the state machine "MQTT job state machine" changes to the state "IDLE". The output parameter "mqttEstablished" is set. The KeepAlive interval is started if necessary. If the "TRCV" block detects an error or the received MQTT control packet is not a "CONNACK" packet, the state machine changes to the state "MQTT_ERROR_STATE".
MQTT_CONNECTED_ STATE	In this state, the function block maintains the state until the MQTT connection or TCP connection is cleared. In the "MQTT_CONNECTED" state, the system checks whether there is a send request for one of the following MQTT control packets: • PUBLISH • SUBSCRIBE • UNSUBSCRIBE When the KeepAlive interval is ending soon, the MQTT control packet "PINGREQ" must be sent. Depending on the outcome of the check, the internal state machine "MQTT job state machine" is set to the appropriate state to execute the desired routine.
MQTT_DISCONNECT_ STATE	If the input parameter "enable" is reset, the MQTT connection is cleared. A "DISCONNECT" packet is assembled for this purpose and then sent to the MQTT broker with the "TSEND" block. 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 state machine" is set to the "TCP_DISCONNECT" state. This also ends the TCP connection.

State	Description
MQTT_ERROR_STATE	If an error occurs in the state machine "MQTT state machine", the state "MQTT_ERROR" is the central point of contact. Here, the required parameters (static variables and output parameters) are set or reset. In addition, the following actions are carried out: • The error message of the MQTT command involved is transferred at the output "status". • The number of the state in which the error occurred is output at the output "statusID". • The state machine returns to the "MQTT_DISCONNECTED_STATE".

3.2.4 State machine "MQTT job state machine":

The state machine "MQTT job state machine" is only run through when the state machine "MQTT state machine" is in the "MQTT_CONNECTED" state. This is because it is decided here from which point the state machine "MQTT job state machine" 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 "MQTT job state machine" contains the following states:

- IDLE
- MQTT_PUBLISH_STATE
- MQTT_PUBLISH_EVAL_STATE
- MQTT_PUBACK_STATE
- MQTT_PUBREC_ACK_STATE
- MQTT_PUBREC_SEND_STATE
- MQTT_PUBCOMP_STATE
- MQTT_SUBSCRIBE_STATE
- MQTT_SUBACK_STATE
- MQTT_UNSUBSCRIBE_STATE
- MQTT_UNSUBACK_STATE
- MQTT_PING
- MQTT_PINGRESP

Table 3-22

State	Description
IDLE	As long as there is no transmission impulse or the KeepAlive interval is not expiring, the state is always "IDLE".
MQTT_PUBLISH_STATE	If a positive edge is detected in the state "MQTT_CONNECTED_STATE" at the input parameter "publishMessage" and no other send job (SUBSCRIBE or UNSUBSCRIBE) is triggered, the internal state machine "MQTT job state machine" is set to the state "MQTT_PUBLISH_STATE". The sender routine starts in the state "MQTT_PUBLISH_STATE" . First, the "PUBLISH" packet with the given parameters, the topic and the message text is assembled and then it is sent to the MQTT broker with the "TSEND" block. Depending on the quality assurance QoS, the state machine changes to another state: • If QoS equals "0", the state machine changes to the state "MQTT_PUBLISH_EVAL_STATE" to terminate the send job. • If QoS equals "1", the state machine changes to the state "MQTT_PUBACK_STATE" in order to receive an acknowledgement from the MQTT broker. • If QoS is equal to "2", this state machine changes to the state "MQTT_PUBREC_ACK_STATE".
MQTT_PUBLISH_EVAL_STATE	If an error occurs while sending the "PUBLISH" packet, the state machine changes to state "IDLE" and the state machine "MQTT state machine" goes back to "MQTT_ERROR_STATE". If the "PUBLISH" packet has been sent successfully, the state machine changes back to the "IDLE" state.

State	Description
MQTT_PUBACK_STATE	If an error occurs while sending the "PUBLISH" packet, the state machine changes to state "IDLE" and the state machine "MQTT state machine" goes back to "MQTT_ERROR_STATE". If the "PUBLISH" packet has been successfully sent, the state is not exited until a message is received at the "TRCV" block. Since the quality assurance QoS is equal to "1", the MQTT client expects a "PUBACK" packet from the MQTT broker to acknowledge the "PUBLISH" packet. It is checked whether the received acknowledgment is a "PUBACK" packet. When the MQTT broker has confirmed receipt of the message, the state machine changes to the state "IDLE". If the "TRCV" block detects an error or the received MQTT control packet is not a "PUBACK" packet, the state machine changes to the state "IDLE" and the state machine "MQTT state machine" changes to the state "MQTT_ERROR_STATE".
MQTT_PUBREC_ACK_STATE	If an error occurs while sending the "PUBLISH" packet, the state machine changes to state "IDLE" and the state machine "MQTT state machine" goes back to "MQTT_ERROR_STATE". If the "PUBLISH" packet has been successfully sent, the state is not exited until a message is received at the "TRCV" block. Since the quality assurance QoS is equal to "2", a double handshake procedure is started. The MQTT client expects a "PUBREC" packet from the MQTT broker to acknowledge the "PUBLISH" packet. It is checked whether the received acknowledgment is a "PUBREC" packet. When the MQTT broker has confirmed receipt of the message, the state machine changes to the state "MQTT_PUBREC_SEND_STATE". If the "TRCV" block detects an error or the received MQTT control packet is not a "PUBREC" packet, the state machine changes to the state "IDLE" and the state machine "MQTT state machine" changes to the state "MQTT_ERROR_STATE".
MQTT_PUBREC_SEND_STATE	After the MQTT client has received the "PUBREC" packet, it is confirmed by the "PUBREL" packet. A "PUBREL" packet is assembled for this purpose and then sent to the MQTT broker with the "TSEND" block. The state machine changes to the "MQTT_PUBCOMP_STATE".

State	Description
MQTT_PUBCOMP_STATE	This state is the last part of the dual handshake procedure at QoS equal to "2". If an error occurs while sending the "PUBREL" packet, the state machine changes to state "IDLE" and the state machine "MQTT state machine" goes back to "MQTT_ERROR_STATE". If the "PUBREL" packet has been successfully sent, the state is not exited until a message is received at the "TRCV" block. The MQTT client expects a "PUBCOMP" packet from the MQTT broker to acknowledge the "PUBREL" packet. It is checked whether the received acknowledgment is a "PUBCOMP" packet. When the MQTT broker has confirmed receipt of the message, the state machine changes to the state "IDLE". The dual handshake procedure is now complete. If the "TRCV" block detects an error or the received MQTT control packet is not a "PUBCOMP" packet, the state machine changes to the state "IDLE" and the state machine "MQTT state machine" changes to the state "MQTT_ERROR_STATE".
MQTT_SUBSCRIBE_STATE	If a positive edge is detected in the state "MQTT_CONNECTED_STATE" at the input parameter "subscribeToTopic" and no other send job (PUBLISH or UNSUBSCRIBE) is triggered, the internal state machine "MQTT job state machine" is set to the state "MQTT_PUBLISH_STATE". The sender routine starts in the state "MQTT_SUBSCRIBE_STATE". First, the "SUBSCRIBE" packet with the given parameters and the topic is assembled and then it is sent to the MQTT broker with the "TSEND" block. The state machine changes to the "MQTT_SUBACK_STATE".

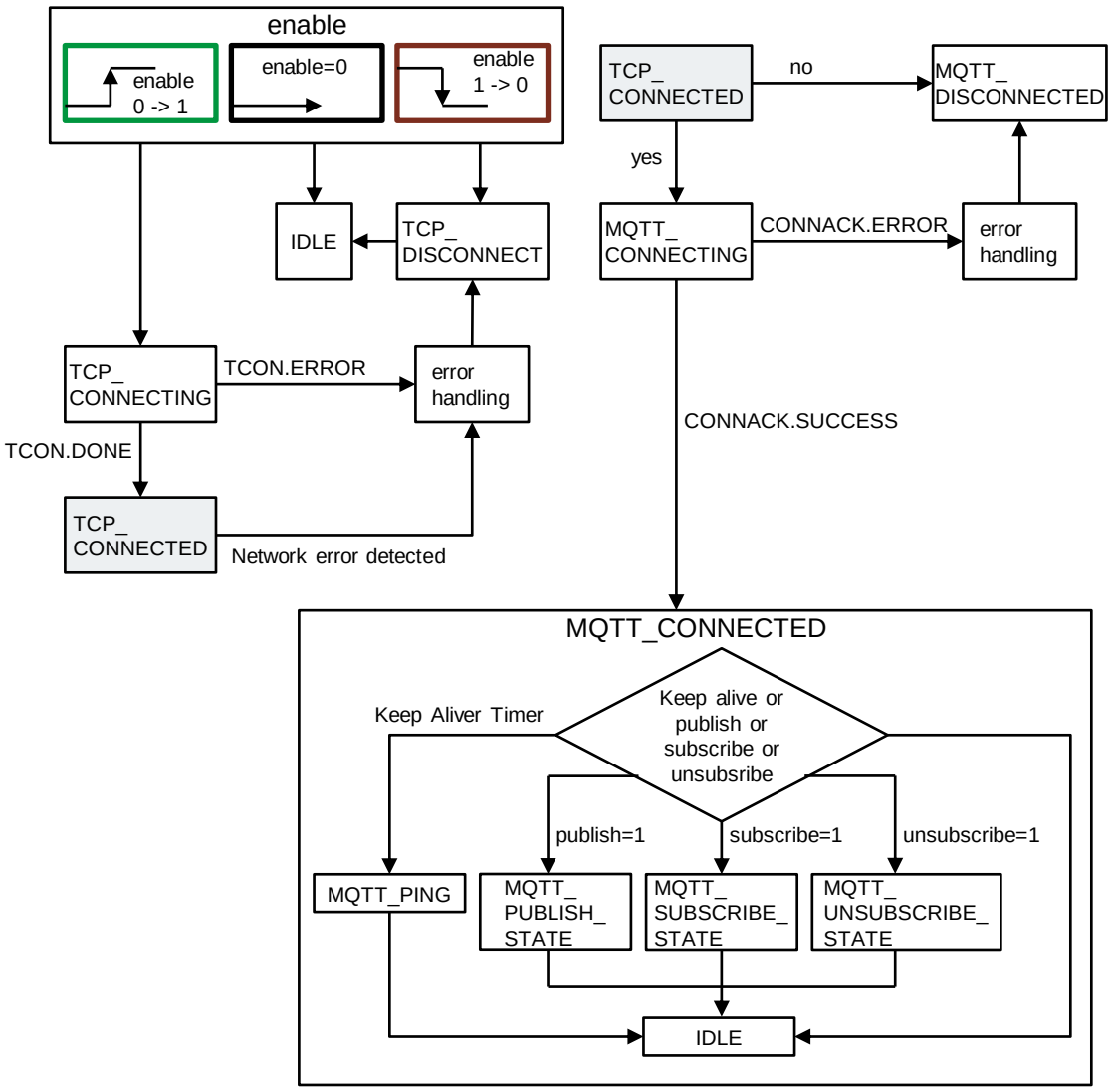
State	Description
MQTT_SUBACK_STATE	If an error occurs while sending the "SUBSCRIBE" packet, the state machine changes to state "IDLE" and the state machine "MQTT state machine" goes back to "MQTT_ERROR_STATE". If the "SUBSCRIBE" packet has been successfully sent, the state is not exited until a message is received at the "TRCV" block. The MQTT client expects a "SUBACK" packet from the MQTT broker to acknowledge the "SUBSCRIBE" packet. It is checked whether the received acknowledgment is a "SUBACK" packet. When the MQTT broker has confirmed receipt of the message, the state machine changes to the state "IDLE". If the "TRCV" block detects an error or the received MQTT control packet is not a "SUBACK" packet, the state machine changes to the state "IDLE" and the state machine "MQTT state machine" changes to the state "MQTT_ERROR_STATE".
MQTT_UNSUBSCRIBE_STATE	If a positive edge is detected in the state "MQTT_CONNECTED_STATE" at the input parameter "subscribeToTopic" and no other send job (PUBLISH or SUBSCRIBE) is triggered, the internal state machine "MQTT job state machine" is set to the state "MQTT_UNSUBSCRIBE_STATE". The sender routine starts in the state "MQTT_UNSUBSCRIBE_STATE". First, the "UNSUBSCRIBE" packet with the given parameters and the topic is assembled and then it is sent to the MQTT broker with the "TSEND" block. The state machine changes to the "MQTT_UNSUBACK_STATE".

State	Description
MQTT_UNSUBACK_STATE	If an error occurs while sending the "UNSUBSCRIBE" packet, the state machine changes to state "IDLE" and the state machine "MQTT state machine" goes back to "MQTT_ERROR_STATE". If the "UNSUBSCRIBE" packet has been successfully sent, the state is not exited until a message is received at the "TRCV" block. The MQTT client expects an "UNSUBACK" packet from the MQTT broker to acknowledge the "UNSUBSCRIBE" packet. It is checked whether the received acknowledgment is a "SUBACK" packet. When the MQTT broker has confirmed receipt of the message, the state machine changes to the state "IDLE". If the "TRCV" block detects an error or the received MQTT control packet is not an "UNSUBACK" packet, the state machine changes to the state "IDLE" and the state machine "MQTT state machine" changes to the state "MQTT_ERROR_STATE".
MQTT_PING	If it is determined in the "MQTT_CONNECTED" state that the KeepAlive interval is expiring, the internal state machine "MQTT job state machine" is set to the "MQTT_PING" state. The ping routine starts in the state "MQTT_PING". First, a "PINGREQ" packet is assembled for this purpose and then sent to the broker with the "TSEND" block. The state machine changes to the "MQTT_PINGRESP".
MQTT_PINGRESP	If an error occurs while sending the "PING" packet, the state machine changes to state "IDLE" and the state machine "MQTT state machine" goes back to "MQTT_ERROR_STATE". If the "PING" packet has been successfully sent, the state is not exited until a message is received at the "TRCV" block. The MQTT client expects a "PINGRESP" packet from the MQTT broker to acknowledge the "PINGREQ" packet. It is checked whether the received acknowledgment is a "PINGRESP" packet. When the MQTT broker has confirmed receipt of the message, the state machine changes to the state "IDLE". The KeepAlive interval is restarted. If the "TRCV" block detects an error or the received MQTT control packet is not a "PINGRESP" packet, the state machine changes to the state "IDLE" and the state machine "MQTT state machine" changes to the state "MQTT_ERROR_STATE".

3.2.5 Function diagram

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

Figure 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:
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:
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:
support.industry.siemens.com/cs/ww/en/sc/2067

4.2 Links and Literature

Table 4-1

No.	Topic
\1\	Siemens Industry Online Support https://support.industry.siemens.com
\2\	Link to the entry page of the application example https://support.industry.siemens.com/cs/ww/en/view/109748872
\3\	MQTT specification http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html

4.3 Change documentation

Table 4-2

Version	Date	Change
V1.0	07/2017	First version
V1.1	08/2018	"LMqttQdn" library added.
V2.0	08/2019	Subscribe mechanism added
V2.1	12/2019	Update to TIA Portal V16