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 under https://www.siemens.com/cert.

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.

Criteria of MQTT

The MQTT protocol is distinguished by the following criteria:

- Lightweight protocol with low transport overhead
- Minimal need for network bandwidth through push mechanism
- Function for re-connection disconnection
- Re-sending messages after disconnection
- Mechanism for notifying interested parties after an unpredicted disconnection of a client
- Simple use and implementation thanks to a small set of commands
- Quality of Serice (QoS level) with different reliability levels for the message delivery
- Optional encryption of messages with TLS
- Authentication of publishers and subscribers with username and password

Applicative implementation

To implement the MQTT protocol in a SIMATIC S7 Controller, the "LMQTT" library offers an adequate solution.

The "LMQTT" library provides a function block for the SIMATIC S7-1500 and SIMATIC S7-1200. The function block "LMQTT_Client" integrates the MQTT Client function and allows you to submit 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

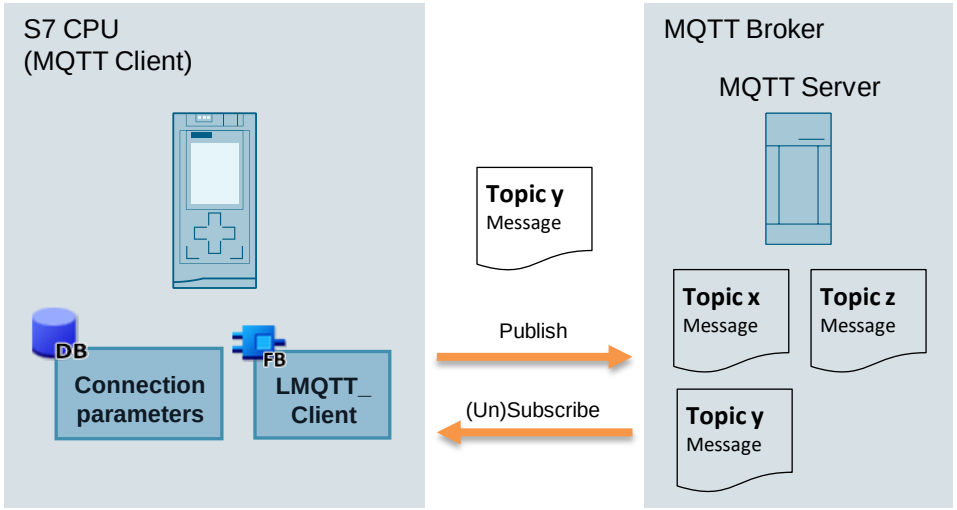


Table 1-1

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 section 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 V4.4 or higher. • Alternatively, you can also use the following components: - CP 1543-1 (6GK7543-1AX00-0XE0) with firmware V2.0 or higher - CP 1545-1 (6GK7545-1GX00-0XE0) - CP 1543SP-1 (6GK7543-6WX00-0XE0) - CP 1243-1 (6GK7243-1BX30-0XE0) with firmware V3.2 or higher - CP 1243-8 IRC (6GK7243-8RX30-0XE0) with firmware V3.2 or higher - CP 1243-7 LTE (6GK7243-7KX30-0XE0 / 6GK7243-7SX30-0XE0) with firmware V3.2 or higher
TIA Portal V16	-	DVD: 6ES7822-1AA06-0YA5 Download: 6ES7822-1AE06-0YA5	-
MQTT Broker	-	-	If you want to encrypt the communication, the MQTT Broker must support TLS.

This application example consists of the following components:

Table 1-3

Components	File name
"LMQTT" library	109780503_Libraries_Comm_Controller_LIB_V1_0_0.zip
This document	109748872_MQTT_Client_DOKU_V3-0_de.pdf

Note With S7-1500 CPUs (firmware V2.0 or higher) or S7-1200 CPUs (firmware V4.4 or higher), you can reach the MQTT Broker via a static IP address or a domain name ("Qualified Domain Name", short: QDN) if you use the "LMQTT" library.

2 Engineering

Note The engineering in this section 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

You can find the module description in the following entry:
https://support.industry.siemens.com/cs/ww/en/view/109780503

2.2 Configuration

The application example in entry 109748872 shows the 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 secured MQTT communication via TLS, you need an S7-1500 CPU with firmware version 2.0 or higher, or an S7-1200 CPU with firmware V4.4 or higher.

2.3 Integration of the Function Block in the User Program

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

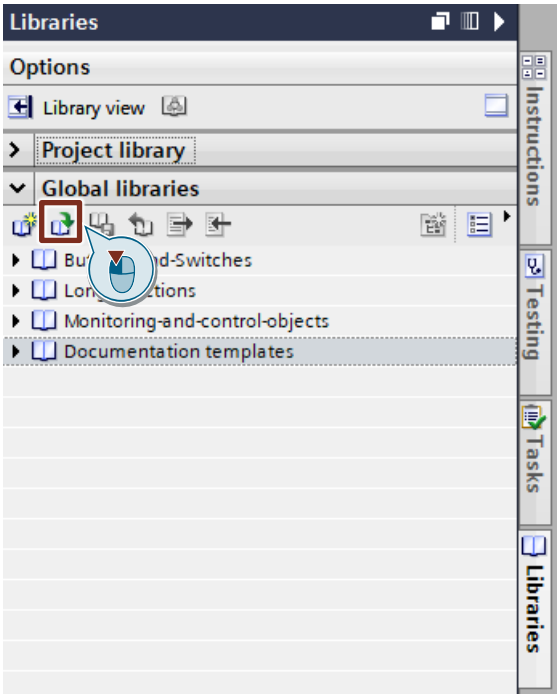
2.3.1 Opening the "LMQTT" Global Library

Note

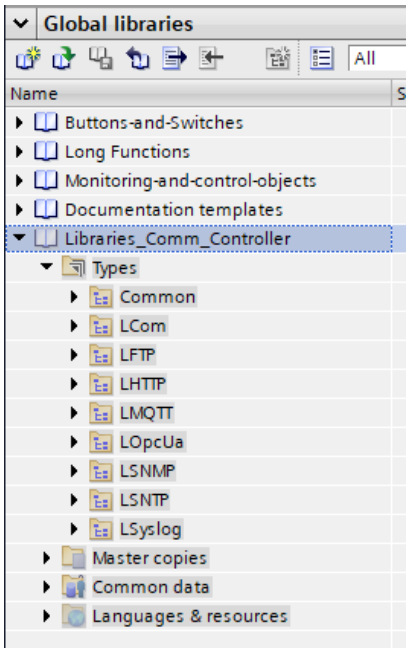
For this section, you must download the "109780503_Libraries_Comm_Controller_LIB_V1_0_0.zip" library and unzip it into a directory of your choice.

The 109780503_Libraries_Comm_Controller_LIB_V1_0_0.zip library can be found in the following entry:
https://support.industry.siemens.com/cs/ww/en/view/109780503

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.

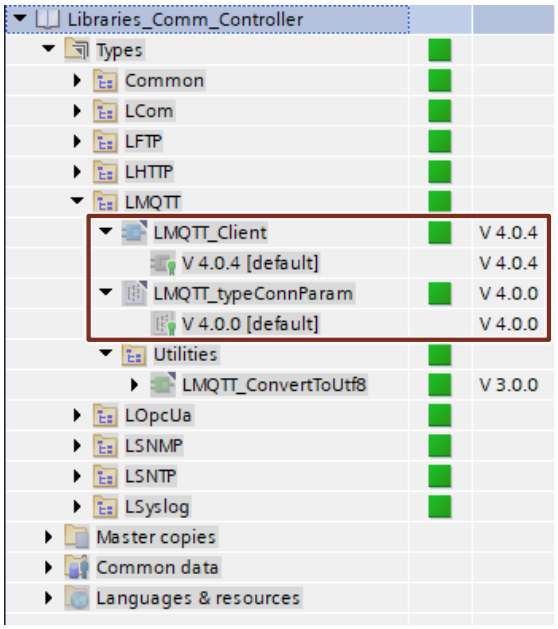


3. Navigate to the folder, into which you unzipped the downloaded file. The folder contains the global library "Libraries_Comm_Controller.al17" ("al" stands for "application library"). Choose this library and click on the "Open" button.
4. The "Libraries_Comm_Controller" library opens and appears under the Global libraries palette.

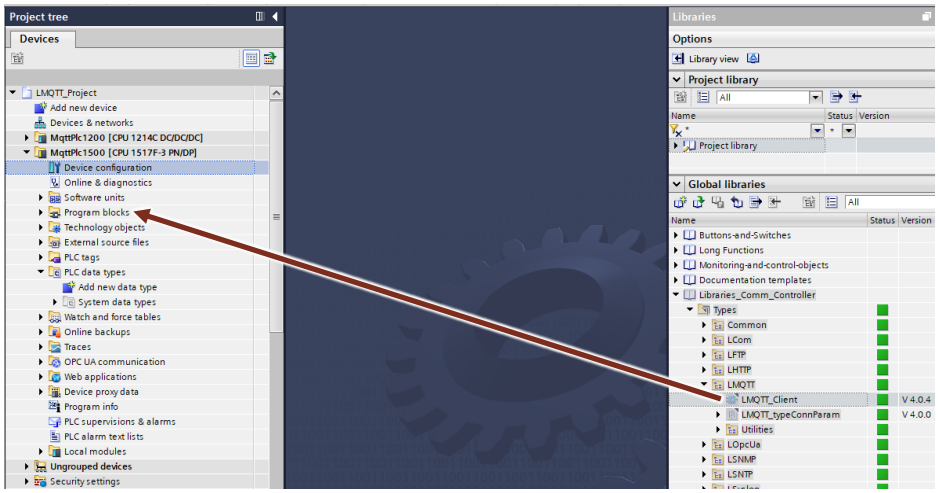


2.3.2 Copying Function Blocks and Data Types to the User Program

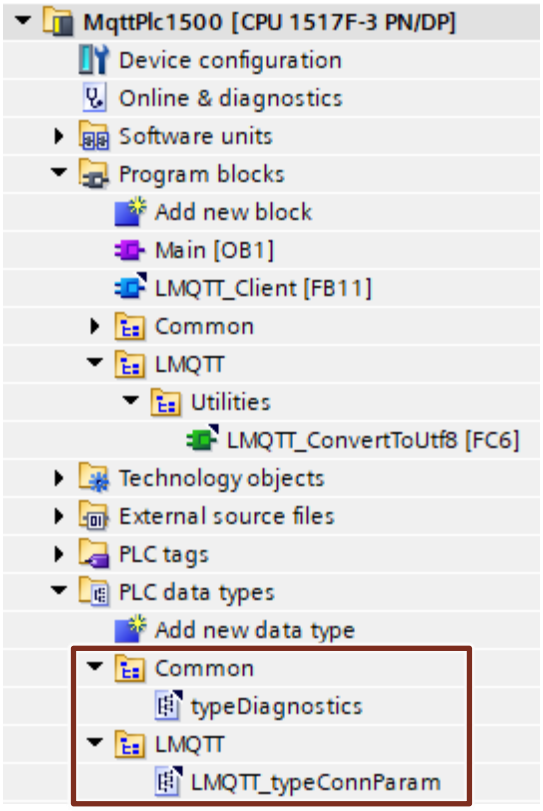
1. In the "Libraries_Comm_Controller" library, you will find the FB "LMQTT_Client" and the corresponding PLC data types under "Types > LMQTT".



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 used by the FB "LMQTT_Client" are automatically inserted into the folder "PLC data types" on your device (e.g., an S7-1500 CPU).

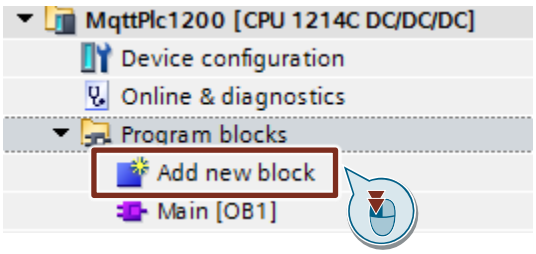


2.3.3 Creating Global Data Block

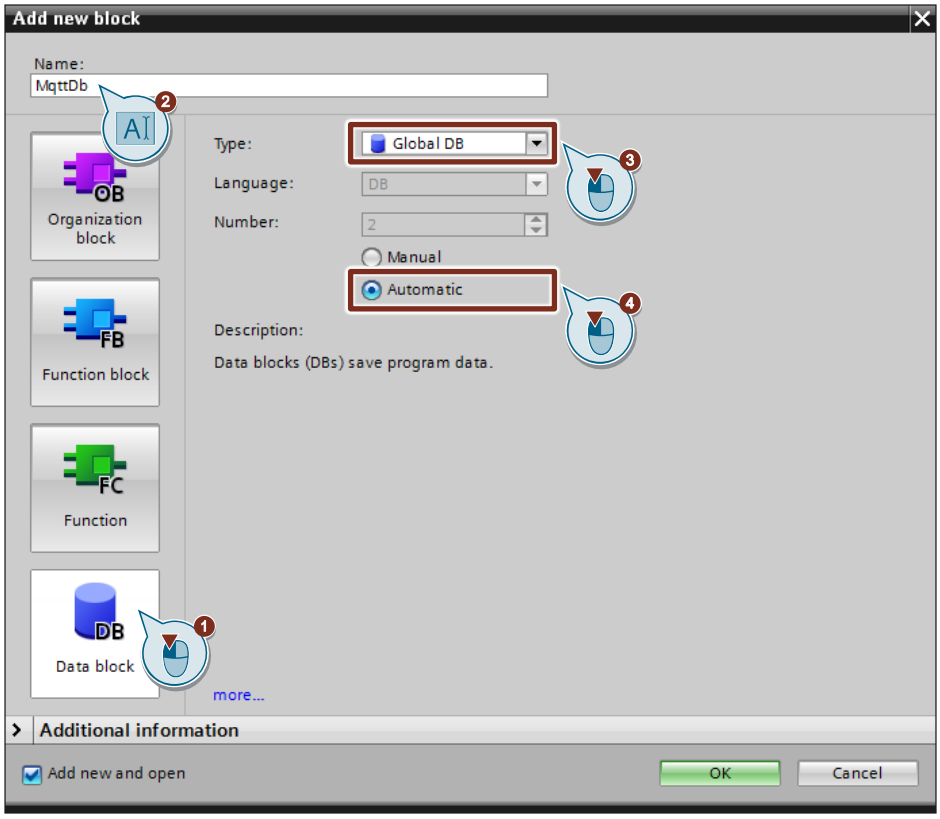
This section shows you how to create a global data block (DB). This DB is used to store the following data:

- 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)

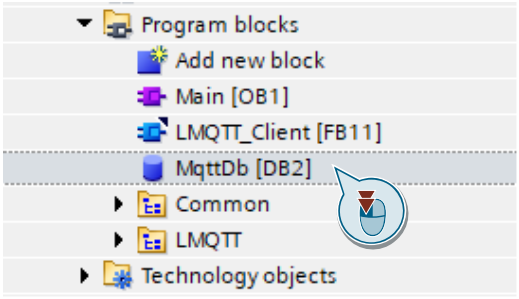
1. Navigate in the "Project tree" to the device folder of the CPU.
2. Open the "Program blocks" folder and double-click the "Add new block" command.
   The dialog "Add new block" opens.



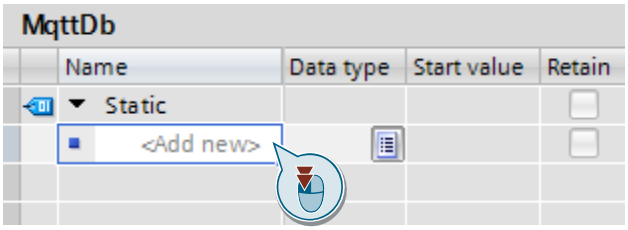
3. Make the following settings and then confirm your entries with the "OK" button.
   - Select the symbol "Data block".
   - Select "Global DB" as the type.
   - Enter the name of the DB.
   - Enable the "Automatic" radio button for automatic number assignment. The number of the global DB is assigned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



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



5. Double-click "<Add new>" to add the corresponding tags.



Result

The following figure shows the tags in the DB "MqttDb" for switching the inputs and outputs of the FB "MQTT_Client".

Figure 2-1

MqttDb

Name	Data type	Start value
Static		
control	Struct	
enable	Bool	false
publish	Bool	false
subscribe	Bool	false
unsubscribe	Bool	false
output	Struct	
valid	Bool	false
done	Bool	false
busy	Bool	false
error	Bool	false
status	Word	16#0
diagnostics	"typeDiagnostics"	
connParam	"LMQTT_typeConnParam"	
clientIdentifier	WString[20]	WSTRING#''
username	WString[20]	WSTRING#''
password	WString[20]	WSTRING#''
willTopic	WString[20]	WSTRING#''
willMsgPayload	Array[0..99] of Byte	
willMsgLen	UInt	0
qos	USInt	0
mqttTopic	WString[100]	WSTRING#''
publishMsgPayload	Array[0..999] of Byte	
publishMsgLen	UDInt	0
retain	Bool	false
receivedTopic	WString[200]	WSTRING#''
receivedMsgPayload	Array[0..999] of Byte	
receivedMsgDataLen	UDInt	0
receivedMsgStatus	USInt	0

The following figure shows the parameters of the tag "connParam".

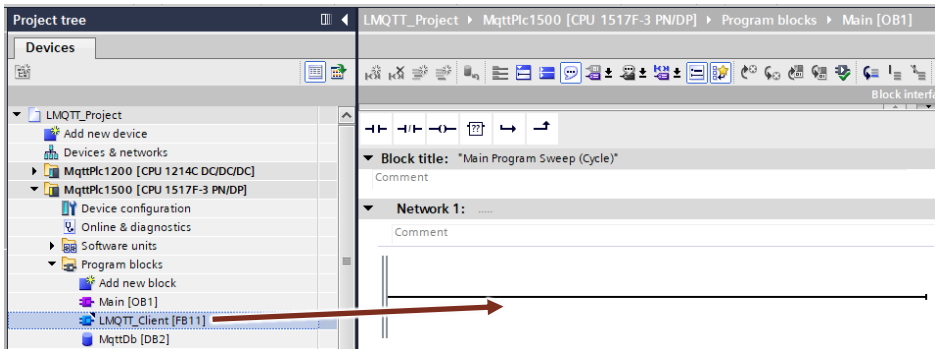
Figure 2-2

MqttDb

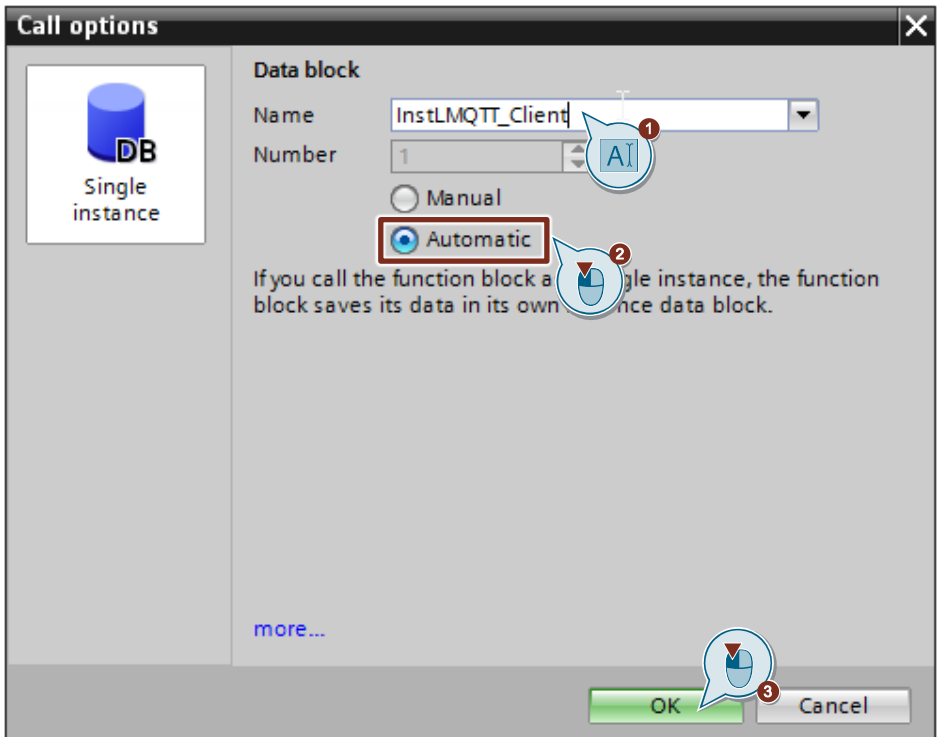
Name	Data type	Start value
connParam	"LMQTT_typeConnParam"	
hwId	HW_ANY	0
connId	CONN_OUC	0
broker	String	''
port	UInt	0
tls	Struct	
enableTls	Bool	false
validateServerIdentity	Bool	false
brokerCert	UDInt	0
clientCert	UDInt	0
keepAlive	UInt	0

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 the block "Main [OB1]" to open the corresponding program editor.
3. Drag & drop the FB "LMQTT_Client" from the project navigation 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.
   - Enable the "Automatic" radio button for automatic number assignment. The number of the instance DB is assigned by the TIA Portal.
   - Click "OK" to confirm the settings.

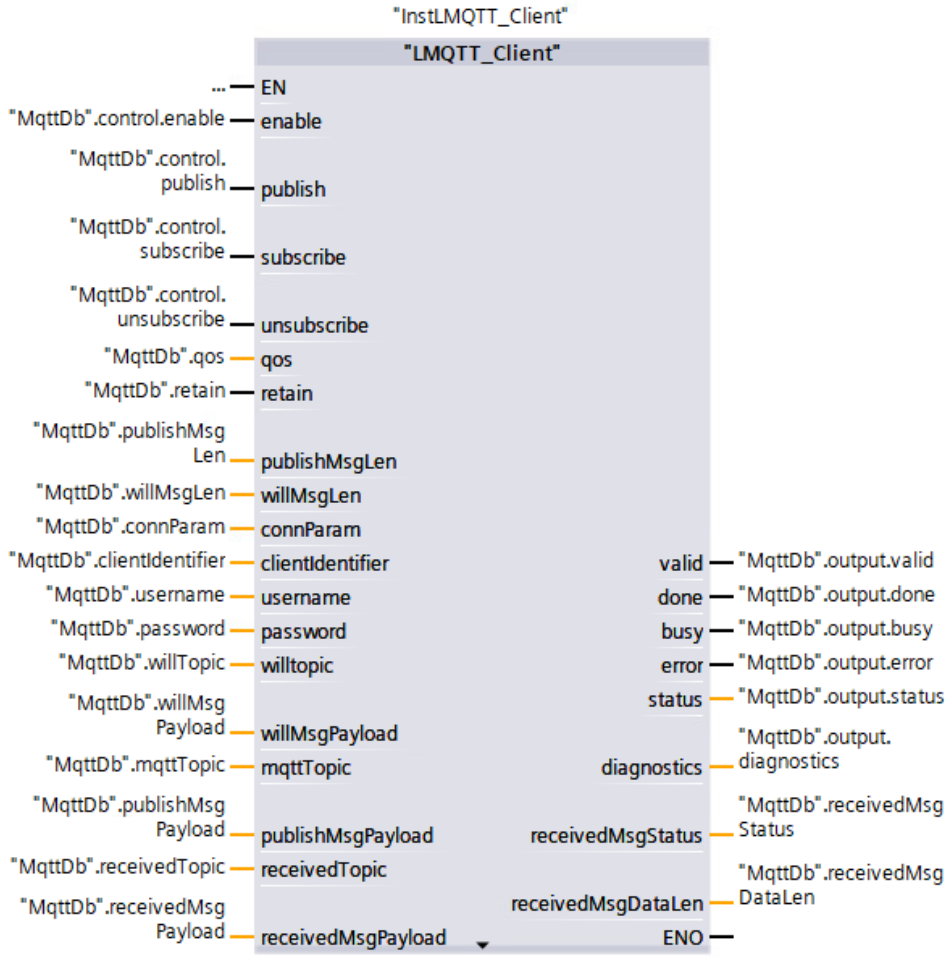


6. Assign the tags that you have created in the global data block to the inputs and outputs of the FB (see section 2.3.3).

Result

The following figure shows the linking of the DB "MqttDb" tags on the FB "MQTT_Client".

Figure 2-3



2.4 Configuration of the Security Feature

Note

You only need to configure the security feature if you want to secure the MQTT communication via TLS.
In the application example, only the MQTT client authenticates the MQTT broker, but not the other way around.

Encryption via TLS is based on certificates. The MQTT client therefore needs the certificate of the MQTT broker in order to authenticate him. Two cases need to be distinguished:

- If the broker uses a self-signed certificate, it needs to be imported into the client during configuration. When the connection is established, the broker then sends his certificate to the client, who compares the received certificate to the one imported before. Both certificates match.
- Alternatively, the broker can use a certificate that is derived from a Certification Authority (CA). In this case, the CA root certificate of the broker must be imported into the client during configuration. When the connection is established, the broker then sends his derived certificate to the client. By using the root certificate, the client can verify the received certificate from the broker.

As the menus and clicks are indentical in both cases, only the import of a "certificate" into the MQTT client is explained.

Requirement for TLS 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 certificate of the broker is available for importing it into the client.
- 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 TIA Portal Global Certificate Manager

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

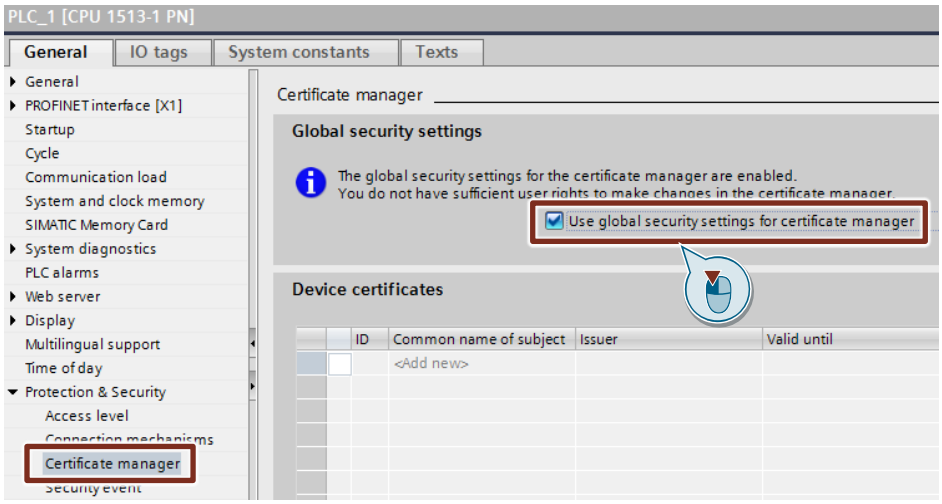
In the TIA Portal, the certificates are managed in the global certificate manager. The certificate manager contains an overview of all certificates used in the project. In the certificate manager, for example, you can import new certificates and export, renew, or replace existing certificates. Each certificate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

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



Result

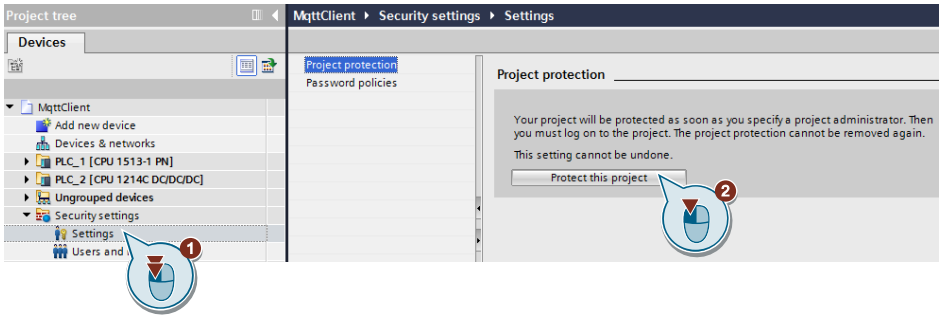
The new entry "Security Settings" appears in the project navigation.

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 the entry "Settings" in the project navigation under "Security settings".
2. The user administration editor opens and the project protection area is displayed.
   Click the "Protect this project" button.



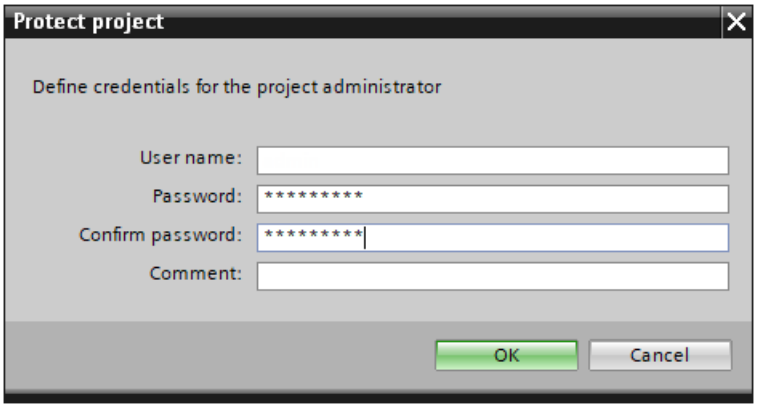
3. This opens the dialog "Protect Project".
   Enter a username and password.

   The password must comply with the following guidelines:

   - Password length: A minimum of eight 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.

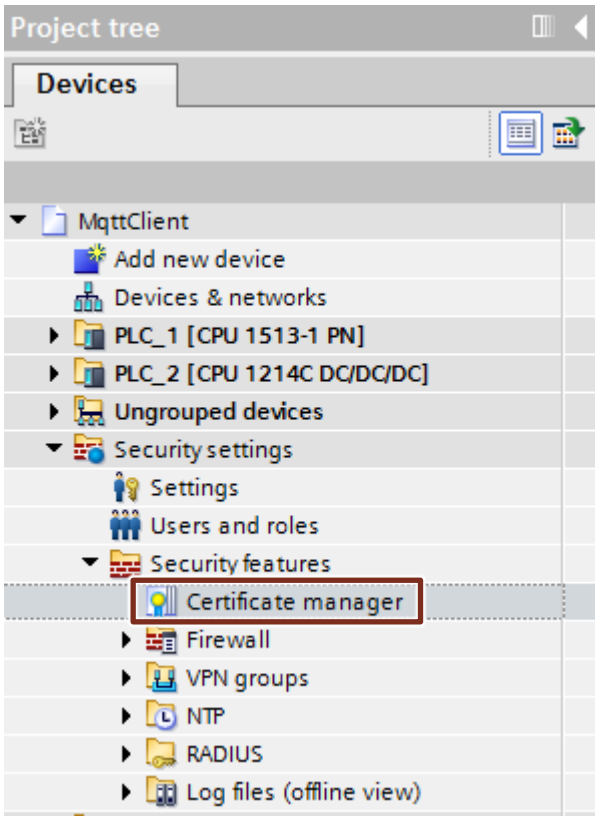
4. You may enter a comment if required.
   Confirm your entries with "OK".



Result

You have activated the user administration. 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 2-4



Using the global certificate manager

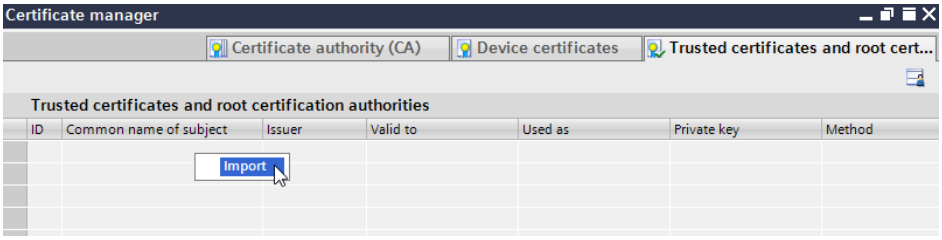
With the global certificate manager, you now have the option of importing third-party certificates into TIA Portal. By double-clicking on the line "Certificate manager" you gain access to all certificates in the project, divided into the following tabs:

- "Certificate Authority (CA)"
- "Device certificates"
- "Trusted certificates and core certification authorities"

1. Double-click 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 core certification authorities".

3. Right-click into the registry and choose "Import" in the context menu.

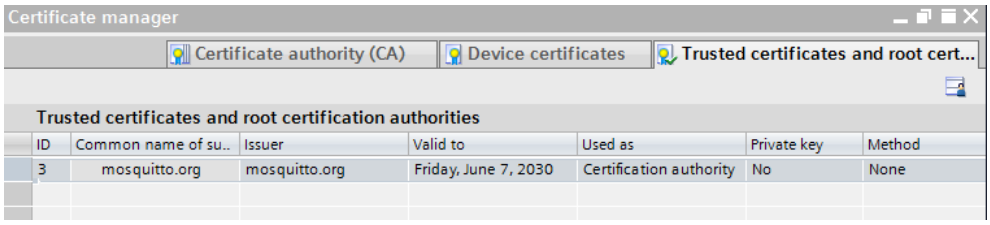


4. In the file explorer, navigate to the certificate's storage folder and import the certificate. Pay attention to its import format:
   - CER, DER, CRT or PEM for certificates without a private key
   - P12 (PKCS12 archive) for certificates with a private key

Result

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

Figure 2-5



Note

If also the MQTT broker shall authenticate the MQTT client, you must import either the self-signed certificate of the client or his CA-certificate into the broker. Pay attention to the following:

- If the client certificate is CA-derived, it has to be the same CA that is used for signing the broker certificate.

Of course, the client certificate muss also be loaded into the client:

- 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 CPU Certificate Manager

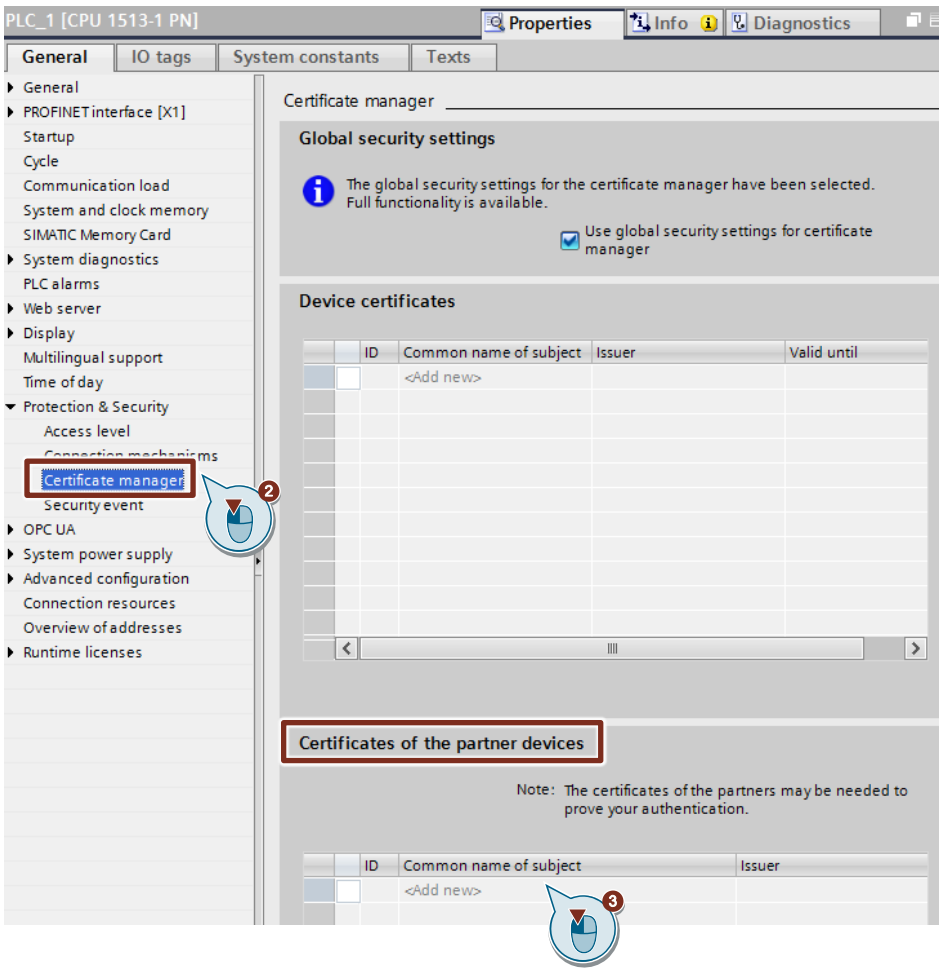
The certificate is currently only located in the global certificate manager of the TIA Portal. Certificates imported via the certificate manager into the global security settings are not automatically assigned to the corresponding modules.

To authenticate the MQTT broker, you have to load the 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.

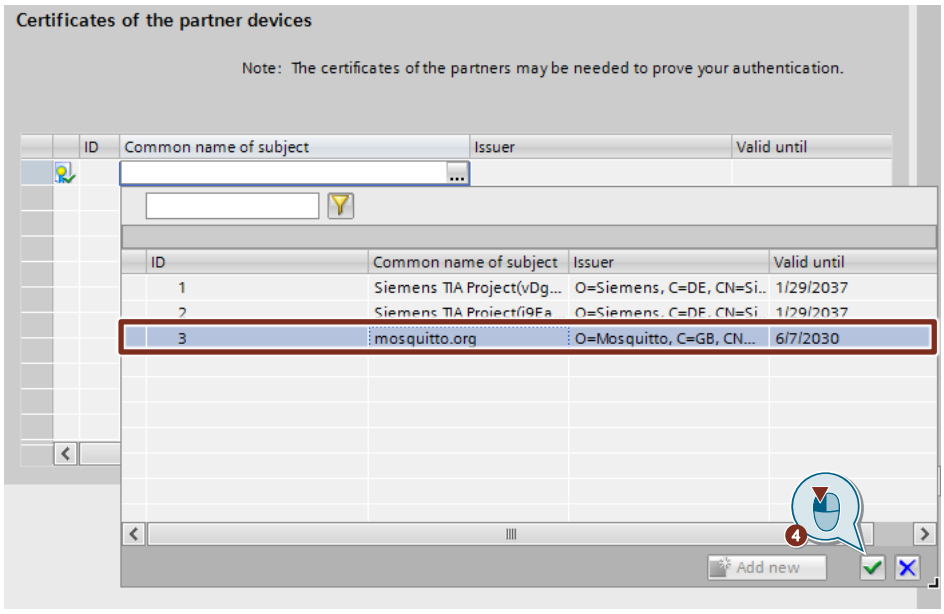
This assignment is made in the local security settings of the module in the entry "Certificate manager" via the table editor "Device certificates". The certificates of the global certificate manager are available for the certificate assignment.

The following steps show you how to assign the 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 Inspector 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 new" in the table of certificates. This inserts a new row into the table.



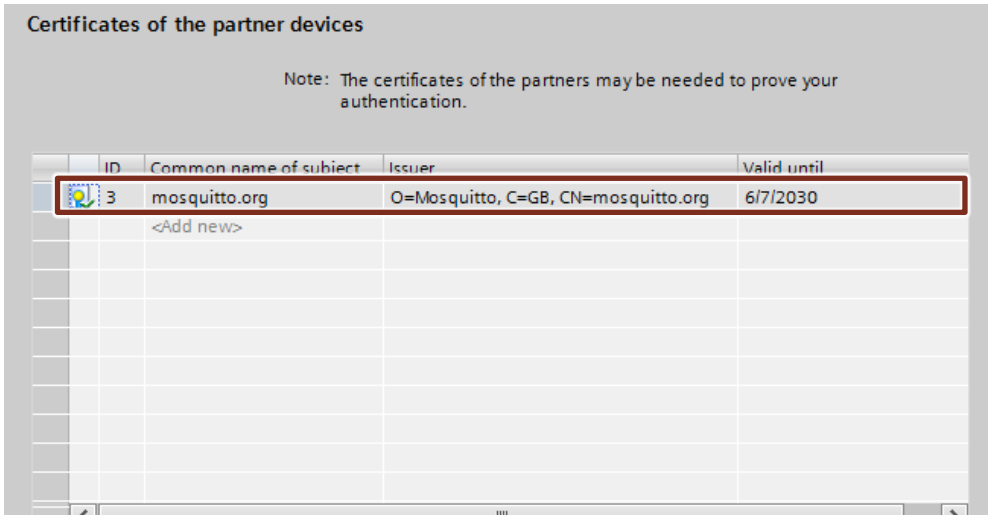
4. Click in the new row. The selection for new certificates opens. Select the previously imported certificate from the global certificate manager and click the green check mark.



Result

The selected certificate was assigned to the CPU and provided with an ID. The ID is the number of the certificate. Enter this value in the connection parameters for the "brokerCert" parameter (see Figure 2-2).

Figure 2-6



Note If also the MQTT broker shall authenticate the MQTT client, the client certificate must also be assigned to the client and shown in the section "Device certificates". Enter the ID of the certificate in the connection parameters for the "clientCert" parameter (see Figure 2-2).

2.5 Parameterization and Operation

Setting the parameters

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

All parameters that you can define yourself are located in the global data block "MqttDb". Above all, set the values of the following parameters in the "Start value" column:

- connection indentifier
- client indentifier
- IPv4 address or domain name of the MQTT Broker. The domain name must end with a ".".
- Port on which the MQTT Broker receives the messages
  - unsecured connection: remote port 1883
  - secured connection: remote port 8883
- Parameters for secure communication
  - Secure connection (TLS) activated
  - ID of the broker certificate (only relevant for a secure connection); the certificate is either self-signed or CA-derived
  - ID of your own client certificate, if the MQTT broker also authenticates the client (only relevant for a secure connection)
- MQTT parameters, e.g.
  - Login information for the MQTT broker
  - Topic
  - Message text

Then load the project into your CPU.

The following table shows the tags in the DB "MqttDb" to interconnect the inputs and outputs of the FB "MQTT_Client".

Table 2-1

Tag		Data type	Note
control		Struct	This data structure contains the tags for controlling the jobs of the FB "LMQTT_Client".
	enable	Bool	This tag controls the connection establishment. • Set the tag to the value "1" to establish the TCP and MQTT connection. • If the value of the tag changes from "1" to "0" (negative edge), the TCP and MQTT connection is disconneted.
	publish	Bool	Use this tag to start a job to send a PUBLISH packet.
	subscribe	Bool	Use this tag to start a job to send a SUBSCRIBE packet.
	unsubscribe	Bool	Use this tag to start a job to send a UNSUBSCRIBE packet.
output		Struct	This data structure contains the tags for evaluating the outputs of the FB "LMQTT_Client".
	valid	Bool	Status display True: The values for the outputs of the FB "LMQTT_Client" are valid.
	done	Bool	Status display • True: Job executed with no errors. • False: Job not yet started or still processing.
	busy	Bool	Status display • True: Job not finished yet. A new job cannot be started. • False: Job not yet started or already finished.
	error	Bool	Status display • True: Error occurred • False: No error
	status	Word	Status of the FB "LMQTT_Client" Detailed information can be found in the library description in entry 109780503.
	diagnostics	"typeDiagnostics"	Diagnostic information of the FB "LMQTT_Client" Detailed information can be found in the library description in entry 109780503.

Tag	Data type	Note
connParam	"LMQTT_typeConnParam"	Enter the parameters to establish a connection to the MQTT Broker. Detailed information can be found in Table 2-2.
clientIdentifier	WString[20]	Enter the Client identifier that is used when establishing the connection (e.g., "SiemensClient").
username	WString[20]	Optionally, it is possible to enter a username for the connection setup. If no username is entered, the parameter is not evaluated.
password	WString[20]	Optionally, it is possible to enter a password for the connection setup. If no password is entered, the parameter is not evaluated.
willTopic	WString[20]	Optionally, it is possible to enter a topic to which the "Last Will" message will be sent.
willMsgPayload	Array[*] of Byte	Optionally, it is possible to enter a message that will be sent as the "Last Will". The length of the array can be chosen as desired. The real message length sent from this array is specified by the tag "willmesscnt".
willMsgLen	UInt	Current length of valid data in the array "willMessage".
qos	USInt	Enter the Quality of Service with which the messages are sent. Possible values are 0, 1, or 2.
mqttTopic	WString[100]	Enter the MQTT topic that will be used in the publish, subscribe, or unsubscribe job.
publishMsgPayload	Array[*] of Byte	Enter the MQTT message that is transmitted as user data in the publish job. The length of the array can be chosen as desired. The real message length sent from this array is specified with the tag "pubMessageCnt".
publishMsgLen	UDInt	Current length of valid data in the array "message".

Tag	Data type	Note
retain	Bool	Enter whether the data is sent with or without the "retain" flag. • True: The data is sent with the "retain" flag. • False: The data is sent without the "retain" flag.
receivedTopic	WString[200]	The MQTT topic on which a message is received regarding the subscription is stored in this tag.
receivedMsgPayload	Array[*] of Byte	This tag stores the user data received in the message via a subscription. The length of the array can be chosen as desired. The real message length received in this array is shown in the tag "receivedMsgLen".
receivedMsgDataLen	UDInt	Number of valid data in the array "receivedmessage".
receivedMsgStatus	USInt	This tag indicates, for one cycle at a time, when a new message has been received (subscription). • 0: No new message received. • 1: New valid message received. • 2: New message received, but message invalid or received data is larger than the memory area of the receive topic or receive message.

The following table shows the parameters of the tag "connParam".

Table 2-2

Parameters	Data type	Note
hwId	HW_ANY	Enter the hardware identifier of the PN/IE interface for establishing the connection. If the tag has the value "0", a suitable hardware identifier is automatically selected.
connId	CONN_OUC	Connection ID for establishing the connection.
broker	String	The IP-address or hostname of the broker
port	UInt	Enter the MQTT port. • port 1883: unsecured connection • port 8883: secured connection
tls	Struct	This data structure contains the parameters for a secure connection.
enableTls	Bool	Enter the value "True" if the connection will be secured with TLS.
validateServerIdentity	Bool	Enter the value "True" if the certificate of the MQTT Broker will be validated when establishing the connection.
brokerCert	UDInt	Certificate ID of the MQTT Broker certificate.
clientCert	UDInt	Certificate ID of the MQTT Client certificate.
keepAlive	UInt	Enter a value in seconds for the activation of the Keep-Alive mechanism of MQTT. If the tag has the value "0", no Keep-Alive is active.

Note If the TCP connection will 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 into 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. For this, trigger the input "enable" of the function block "LMQTT_Client". As long as the input "enable" is set to "True", the connection is maintained. If the input "enable" is reset to "False", the connection is disconnected.

In the positive case, the internal state machines will loop through and establish a TCP and MQTT connection to the MQTT Broker. The output tag "status" is set to the value "16#7004" and signals an existing TCP and MQTT connection.

Now you can perform the following functions:

- Send MQTT message: Trigger the input tag "publish".
- MQTT message received for a subscribed topic: Trigger the input tag "subscribe".
  If the connection to the MQTT Broker is interrupted (status = 16#9000), the connection is automatically re-established. After a disconnection, it is necessary to perform a "subscribe" job for the subscribed topics.
- Unsubscribe yourself from subscribed topics: Trigger the tag "unsubscribe".

If the connection to the MQTT Broker is not established, check the output tag "status" and "diagnostics" to diagnose the error.

2.6 Error Handling

For information on the meaning of the values of the tags "status" and "diagnostics", see the library description in the following entry:

https://support.industry.siemens.com/cs/ww/en/view/109780503

3 Useful Information

3.1 Fundamentals of MQTT

Note A detailed description of MQTT can be found in the MQTT specification description (see \3\ in section 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 criterion for "Quality of Service"
- 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:

- Logging into the MQTT Broker
- Publish mechanism
- Subscribe and unsubscribe mechanisms
- 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.

Keep-Alive

MQTT supports the "Keep-Alive" feature. This ensures that the connection is still open and the MQTT Client and MQTT Broker are connected.

For the Keep-Alive, 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 Keep-Alive 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 Keep-Alive 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 Keep-Alive interval begins again.

Note

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

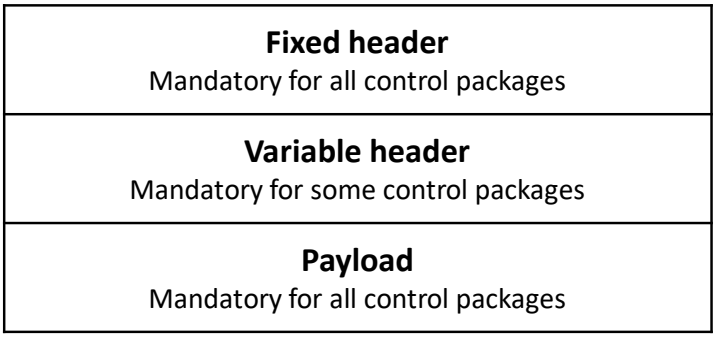
The first time an MQTT Client subscribes to a topic, it usually gets a message only when another MQTT Client sends a message with the subscribed topic the next time. With "Retained messages", the subscriber receives the last value sent to the topic prior to its subscription request, delivered immediately.

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 creates a job for the MQTT Broker. The broker confirms the request depending on the job.

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 section 4.2).

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 Keep-Alive 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	User name 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" = "variable 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" = "variable 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" = "variable 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" = "variable 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" = "variable 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 Keep-Alive interval is greater than "0", the Keep-Alive function is active. If the Keep-Alive function is active, the MQTT Client must send at least one message to the MQTT Broker within the Keep-Alive 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 Keep-Alive 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 Keep-Alive function. The Keep-Alive 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 FB "LMQTT_Client" Works

3.2.1 Requirements and Implementation

The following requirements must be fulfilled for a communication relation between an MQTT Client and an MQTT Broker:

1. The TCP connection to the MQTT Broker is successfully established (Output "status" = 16#7003 "STATUS_MQTT_CONNECTING").
2. The function block "LMQTT_Client" has logged into the broker as the MQTT Client via the existing TCP connection and established a connection (Output "status" = 16#7004 "STATUS_MQTT_CONNECTED").
3. The trigger to send the message or to receive the MQTT connection ("Keep-Alive") 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 Keep-Alive 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 "STATE_MACHINE_FUNCTION_BLOCK_TCP": Management of the TCP connection
- State machine "MQTT_STATE_MACHINE": Management of the MQTT connection, the sending and receiving process
- State machine "MQTT_COMMANDS": MQTT Control Package Management

3.2.2 State Machine "STATE_MACHINE_FUNCION_BLOCK_TCP"

The state machine "STATE_MACHINE_FUNCTION_BLOCK_TCP" is started when a positive edge is detected at 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).
- It controls the breaking of the TCP and MQTT connection.
- If an error has occurred or no positive edge was detected at the "enable" input parameter, it sets all static tags and the other state machines to a defined state.

The state machine "STATE_MACHINE_FUNCTION_BLOCK_TCP" contains the following states:

- FB_STATE_NO_PROCESSING
- FB_STATE_VALIDATE_INPUT
- FB_STATE_TCP_CONNECTING
- FB_STATE_OPERATING_MONITOR_TCP
- FB_STATE_RECONNECTING
- FB_STATE_DISABLING

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

Table 3-20

State	Description
FB_STATE_ NO_PROCCESSING	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 detected at input parameter "enable", the state machine is set to the state "FB_STATE_VALIDATE_INPUT" and the output "valid" is set to the value "TRUE".
FB_STATE_ VALIDATE_INPUT	In this state, all connection parameters are read in and evaluated. The FB changes to the state "FB_STATE_TCP_CONNECTING" without a switching condition.
FB_STATE_ TCP_CONNECTING	The TCP connection to the MQTT Broker is established in this state. If the TCP connection is successfully established with "TSEND_C", the FB changes to the state "FB_STATE_OPERATING_MONITOR_TCP" and the value "16#7003" is output to "status". The TCP connection remains established until it is disconnected with "TSEND_C". If an error occurs when establishing the connection, the status messages for the error are output at the "status" and "diagnostics" outputs, and the FB changes to the state "FB_STATE_NO_PROCESSING".
FB_STATE_OPERATING_ MONITOR_TCP	In this state, the following actions are performed: • Receive and evaluate MQTT control packet • Send MQTT control packets • The state of the TCP connection is monitored. If the TCP connection is interrupted (e.g., due to a broken cable), the FB changes to the state "FB_STATE_RECONNECTING". • Manage state machine "MQTT_STATE_MACHINE" (see section 3.2.3)
FB_STATE_RECONNECTING	When the TCP connection is re-established, the FB changes back to the state "FB_STATE_OPERATING_MONITORING_TCP".
FB_STATE_DISABLING	In this state, the MQTT and TCP connection is disonnected.

3.2.3 State Machine "MQTT_STATE_MACHINE"

The state machine "MQTT_STATE_MACHINE" is started automatically when the TCP connection is established and the value "7003" is output to "status". This state machine has the following functions:

- "Handshake procedure" for establishing the MQTT connection.
- The inputs "publish", "subscribe", and "unsubscribe" are evaluated in order to send the MQTT control palette that matches the job.
- Managing the "MQTT_COMMANDS" state machine.
- Sending PING packet before the Keep-Alive interval expires.

The state machine "MQTT_STATE_MACHINE" contains the following states

- MQTT_CONNECT_STATE_NO_PROCESSING
- MQTT_CONNECT_STATE_BUILD_PAKET
- MQTT_CONNECT_STATE_SEND_PAKET_WAIT_FOR_CONNACK
- MQTT_CONNECT_STATE_CONNECTED

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

Table 3-21

State	Description
MQTT_CONNECT_ STATE_NO_PROCESSING	If the TCP connection is not established or if an error occurs, the state machine is in the "MQTT_CONNECT_NO_PROCESSING" state. Only when a TCP connection is established is the switching condition to the "MQTT_CONNECT_STATE_BUILD_PAKET" state automatically activated.
MQTT_CONNECT_ STATE_BUILD_PAKET	The MQTT connection to the MQTT Broker is established in this state. A "CONNECT" packet is assembled for this purpose and then sent to the MQTT Broker with the "TSEND_C" block. The state machine changes to the state "MQTT_CONNECT_STATE_SEND_PAKET_WAIT_FOR_ CONNACK".
MQTT_CONNECT_ STATE_SEND_PAKET_ WAIT_FOR_CONNACK	The MQTT Client expects a "CONNACK" packet from the MQTT Broker to acknowledge the "CONNECT" packet. When the MQTT Broker has confirmed receipt of the "CONNECT" packet with a "CONNACK" packet, the output "status" is set to the value "16#7004". The status display "16#7004" indicates that the MQTT connection is established.
MQTT_CONNECT_ STATE_CONNECTED	In this state, the following actions are performed: • A check is conducted to see whether there is a send job for one of the following MQTT control packets: - PUBLISH - SUBSCRIBE - UNSUBSCRIBE • Manage state machine "MQTT_COMMANDS" (see section 3.2.4) • Monitor Keep-Alive interval. If the Keep-Alive interval is about to end, the MQTT control packet "PINGREQ" must be sent and the Keep-Alive interval must be restarted.

3.2.4 State Machine "MQTT_COMMANDS"

The state machine "MQTT_COMMANDS" is only processed if the state machine "MQTT_STATE_MACHINE" is in the state "MQTT_CONNECT_STATE_CONNECTED". This is because the point the state machine "MQTT_COMMANDS" is started from is decided here. If there is a send impulse for a MQTT message, then the send routine becomes active. After each "publish", "subscribe", or "unsubscribe" send job, the output "done" of the FB "LMQTT_Client" is set to the value "True" and the value "16#0000" is indicated at the "status" output for 1 cycle. Subsequently, the value of the output "status" changes to the value "16#7004". Only when the input "publish", "subscribe", or "unsubscribe" is reset to the value "false", is the output "done" of the FB "LMQTT_Client" also reset to the value "false".

If the Keep-Alive time is ending soon, the PING routine starts.

The state machine "MQTT_COMMANDS" contains the following states:

- MQTT_COMMAND_NO_PROCESSING
- MQTT_COMMAND_STATE_BUILD_PUBLISH
- MQTT_COMMAND_STATE_SEND_PUBLISH
- MQTT_COMMAND_STATE_BUILD_SUBSCRIBE
- MQTT_COMMAND_STATE_SEND_SUBSCRIBE
- MQTT_COMMAND_STATE_BUILD_UNSUBSCRIBE
- MQTT_COMMAND_STATE_SEND_UNSUBSCRIBE
- MQTT_COMMAND_STATE_SEND_PING
- MQTT_COMMAND_STATE_PING_RESP

Table 3-22

State	Description
MQTT_COMMAND_ NO_PROCESSING	As long as there is no send trigger and the Keep-Alive interval does not expire, the state is always "MQTT_COMMAND_NO_PROCESSING".
MQTT_COMMAND_STATE_ BUILD_PUBLISH	If a positive edge is detected at the "publish" input in the "MQTT_CONNECT_STATE_CONNECTED" state, the internal state machine "MQTT_COMMANDS" is set to the state "MQTT_COMMAND_STATE_BUILD_PUBLISH". If no other send job is running, a "PUBLISH" packet or a "PUBREL" packet is assembled and then sent to the MQTT Broker with the block "TSEND_C". The output "status" is set to the value "16#7006" to signal that the MQTT push mechanism is running. The state machine changes to the state "MQTT_COMMAND_STATE_SEND_PUBLISH".
MQTT_COMMAND_STATE_ SEND_PUBLISH	Depending on the QoS level, the MQTT Client may expect one of the following MQTT control packets as acknowledgement of the PUBLISH packet. • QoS = 0 (dec): The send job ends here. • QoS = 1 (dec): "PUBACK" packet". When the MQTT Broker has confirmed receipt of the "PUBLISH" packet with a "PUBACK" packet, the send job is finished. • QoS = 2 (dec): "PUBREC" packet. When the MQTT Broker has acknowledged receipt of the "PUBLISH" packet with a "PUBREC" packet, the MQTT Client sends a "PUBREL" packet as confirmation. The MQTT Client expects a "PUBCOMP" packet from the MQTT Broker to acknowledge the "PUBREL" packet. When the MQTT Broker has confirmed receipt of the "PUBREL" packet with a "PUBCOMP" packet, the send job is finished. When the send job is completed, the following actions are performed: • The output "status" is set to the value "16#0000" for 1 cycle before it is set again to the value "16#7004". The status display "16#7004" indicates that the MQTT connection is established and no job is active. • The output "done" is set to the value "true". Only when the input "publish" is reset to the value "false", is the output "done" of the FB "LMQTT_Client" also reset to the value "false". • The state machine changes to the state "MQTT_COMMAND_STATE_NO_ PROCESSING".

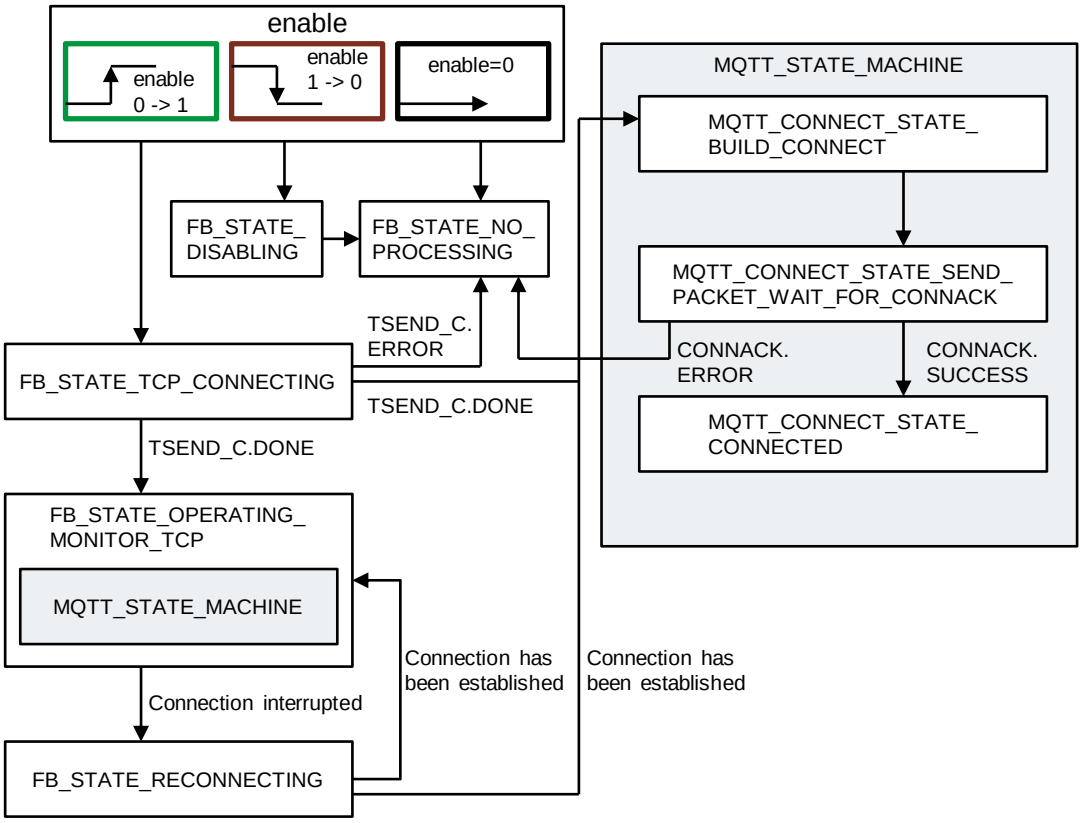
State	Description
MQTT_COMMAND_STATE_ BUILD_SUBSCRIBE	If a positive edge is detected at the "subscribe" input in the "MQTT_CONNECT_STATE_CONNECTED" state, the internal state machine "MQTT_COMMANDS" is set to the state "MQTT_COMMAND_STATE_BUILD_SUBSCRIBE". If no other send job is running, a "SUBSCRIBE" packet is assembled and then sent to the MQTT Broker with the block "TSEND_C". The output "status" is set to the value "16#7008" to signal that the MQTT sub-mechanism is running. The state machine changes to the state "MQTT_COMMAND_STATE_SEND_SUBSCRIBE".
MQTT_COMMAND_STATE_ SEND_SUBSCRIBE	The MQTT Client expects a "SUBACK" packet from the MQTT Broker to acknowledge the "SUBSCRIBE" packet. When the MQTT Broker has acknowledged receipt of the "SUBSCRIBE" packet with a "SUBACK" packet, the following actions are performed: • The output "status" is set to the value "16#0000" for 1 cycle before it is set again to the value "16#7004". The status display "16#7004" indicates that the MQTT connection is established and no job is active. • The output "done" is set to the value "true". Only when the input "subscribe" is reset to the value "false", is the output "done" of the FB "LMQTT_Client" also reset to the value "false". • The state machine changes to the state "MQTT_COMMAND_STATE_NO_ PROCESSING".
MQTT_COMMAND_STATE_ BUILD_UNSUBSCRIBE	If a positive edge is detected at the "unsubscribe" input in the "MQTT_CONNECT_STATE_CONNECTED" state, the internal state machine "MQTT_COMMANDS" is set to the state "MQTT_COMMAND_STATE_BUILD_ UNSUBSCRIBE". If no other send job is running, an "UNSUBSCRIBE" packet is assembled and then sent to the MQTT Broker with the block "TSEND_C". The output "status" is set to the value "16#7010" to signal that the MQTT sub-mechanism is running. The state machine changes to the state "MQTT_COMMAND_STATE_SEND_ UNSUBSCRIBE".

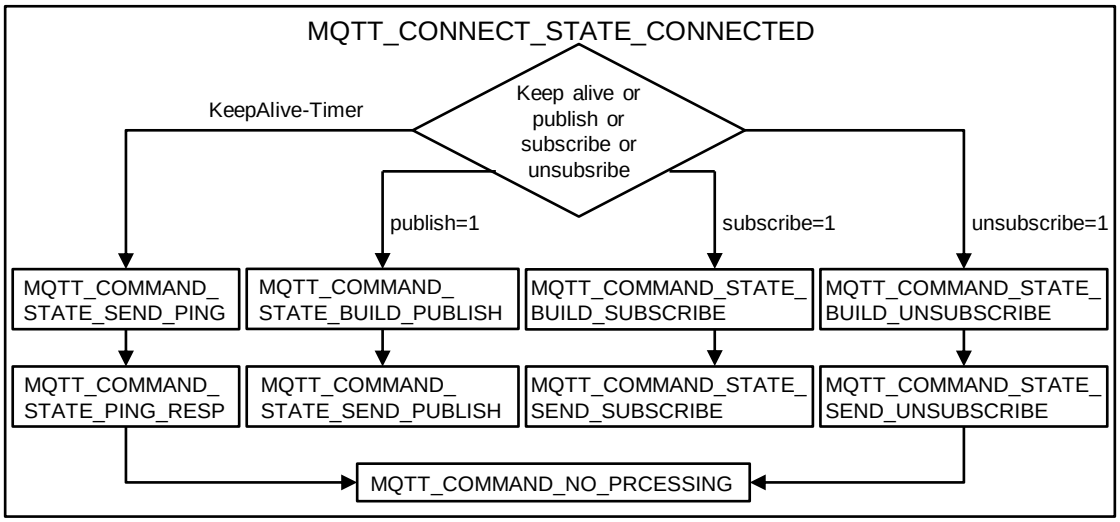
State	Description
MQTT_COMMAND_STATE_ SEND_UNSUBSCRIBE	The MQTT Client expects an "UNSUBACK" packet from the MQTT Broker to acknowledge the "UNSUBSCRIBE" packet. When the MQTT Broker has acknowledged receipt of the "UNSUBSCRIBE" packet with a "UNSUBACK" packet, the following actions are performed: • The output "status" is set to the value "16#0000" for 1 cycle before it is set again to the value "16#7004". The status display "16#7004" indicates that the MQTT connection is established and no job is active. • The output "done" is set to the value "true". Only when the input "unsubscribe" is reset to the value "false", is the output "done" of the FB "LMQTT_Client" also reset to the value "false". • The state machine changes to the state "MQTT_COMMAND_STATE_NO_ PROCESSING".
MQTT_COMMAND_STATE_ SEND_PING	If the Keep-Alive interval has expired in the state "MQTT_CONNECT_STATE_CONNECTED", the internal state machine "MQTT_COMMANDS" is set to the state "MQTT_COMMAND_STATE_SEND_PING" and the Keep-Alive interval is restarted. If no other send job is running, a "PING" packet is assembled and then sent to the MQTT Broker with the block "TSEND_C". The output "status" is set to the value "16#7005" to signal that the MQTT ping mechanism is running. The state machine changes to the state "MQTT_COMMAND_STATE_PING_RESP".
MQTT_COMMAND_STATE_ PING_RESP	The MQTT Client expects a "PINGRESP" packet" from the MQTT Broker as an acknowledgement to the "PING" packet. When the MQTT Broker has confirmed receipt of the "PING" packet with a "PINGRESP" packet, the state machine changes to the state "MQTT_COMMAND_STATE_NO_PROCESSING" and the "status" output is again set to the value "16#7004". The status display "16#7004" indicates that the MQTT connection is established.

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

SiePortal

The integrated platform for product selection, purchasing and support - and connection of Industry Mall and Online support. The SiePortal home page replaces the previous home pages of the Industry Mall and the Online Support Portal (SIOS) and combines them.

- Products & Services
  In Products & Services, you can find all our offerings as previously available in Mall Catalog.
- Support
  In Support, you can find all information helpful for resolving technical issues with our products.
- mySieportal
  mySiePortal collects all your personal data and processes, from your account to current orders, service requests and more. You can only see the full range of functions here after you have logged in.

You can access SiePortal via this address: sieportal.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:
support.industry.siemens.com/cs/my/src

SITRAIN – Digital Industry Academy

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:
siemens.com/sitrain

Industry Online Support app

You will receive optimum support wherever you are with the "Industry Online Support" app. The app is available for iOS and Android:

4.2 Links and literature

Table 4-1

No.	Subject
\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
V3.0	03/2021	FB "LMQTT_Client" V3.0 integrated in libraries for communication for SIMATIC Controllers and documentation for FB "LMQTT_Client" V3.0 adapted
V3.1	04/2024	Update of FB "LMQTT_Client" V4.0.4 and update to TIA Portal V17