SIEMENS

Ingenuity for life

MQTT Publisher for SIMATIC S7-1500

Blocks for S7-1500, Version 1.1

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

1 Introduction

1.1 Overview

Motivation

The unstoppably progressing digitalization has a strong impact on economy and society. The “Internet of Things” (IoT) is one of the main drivers of the digitalization. The term “Internet of Things” stands for one of the greatest current dynamics of change: The increasing interconnection and automation of devices, machines and products.

The “Message Queue Telemetry Transport” protocol (Short form: MQTT) is used as communication protocol in the “Internet of Things”. Due to its light-weight approach, it opens up entirely new possibilities in automation.

Lean and fast: MQTT

The MQTT is a simply structured binary publish and subscribe protocol on TCP/IP level. It is suitable for messaging between devices with minimum functionality and for the transmission via unreliable networks with low bandwidth and high latency. With these characteristics, MQTT plays a vital role for the IoT and in M2M communication.

Characteristics of MQTT

The MQTT protocol stands out with the following characteristics:

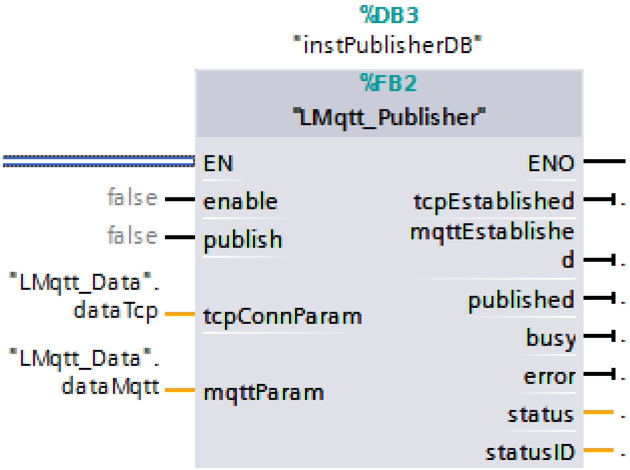
- Light-weight protocol with low transport overhead
- Minimum requirement for network bandwidth thanks to push mechanism
- Function for re-connecting after disconnection
- Re-sending messages after disconnection
- Mechanism for notifying interested parties after a client has disconnected ungracefully
- Simple use and implementation thanks to a small set of commands
- Quality of Service (QoS level) with different reliability levels for the message delivery
- Optional encryption of messages via SSL/TLS
- Authentication of publishers and subscribers via user name and password

Application implementation

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

The application example provides you with a function block for the SIMATIC S7-1500. The “LMqtt_Publisher” function block integrates the MQTT client function and allows you to transmit MQTT messages to a broker (publisher role). During this, the communication can be secured via a TLS connection.

Figure 1-1



%DB3
"instPublisherDB"
%FB2
"LMqtt_Publisher"

Input value	Input	Output
	EN	ENO
false	enable	tcpEstablished
false	publish	mqttEstablished
"LMqtt_Data".dataTcp	tcpConnParam	published
		busy
		error
"LMqtt_Data".dataMqtt	mqttParam	status
		statusID

Note The MQTT client supports the MQTT protocol version 3.1.

1.2 Mode of operation

Diagrammatic representation

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

Figure 1-2

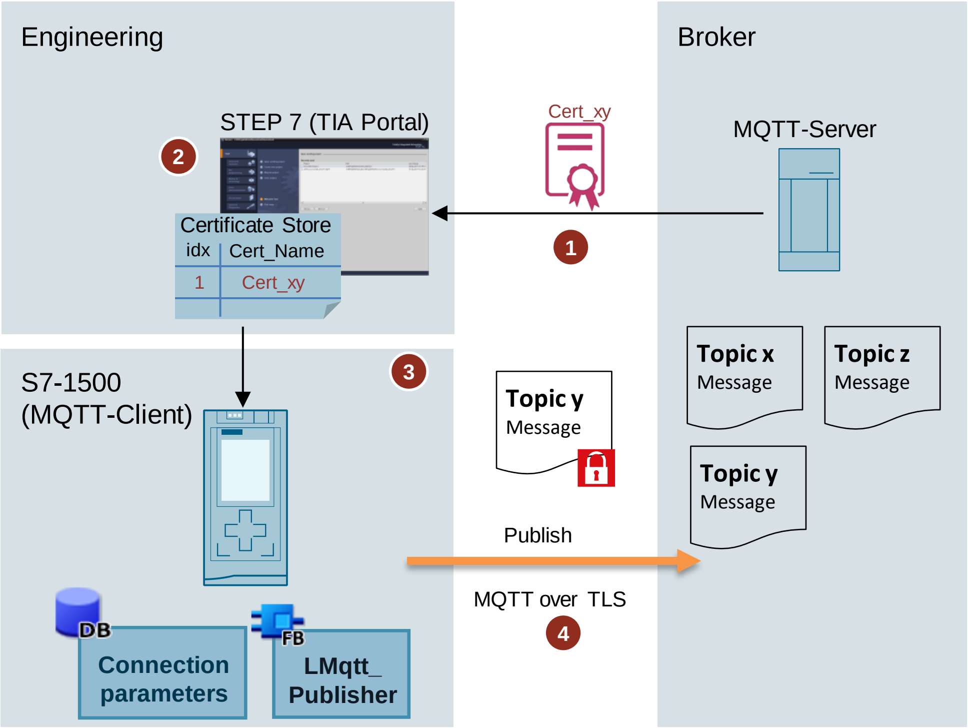


Table 1-1

Step	Description
1	Identifying the CA certificate of the MQTT broker.
2	Importing the external certificate into STEP 7 (TIA Portal). The certificate is now located in the global certificate manager of STEP 7.
3	You need to assign the imported certificate to the S7-1500 station. For the certificate to be recognized as valid, the S7 CPU’s time of day needs to be set to the current time.
4	The “LMqtt_Publisher” function block assumes the role of the publisher and transmits MQTT messages to the broker. The MQTT message is encrypted via a secured connection (MQTT over TLS).

Note A detailed function description of the “LMqtt_Publisher” function block as well as information on the MQTT protocol can be found in chapter 3.

1.3 Components used

This application example has been created with the following hardware and software components:

Table 1-2

Component	Number	Article number	Note
CPU 1513-1 PN	1	6ES7513-1AL01-0AB0	You can also use a different CPU. A secured MQTT communication via TLS requires at least firmware version 2.0.
TIA Portal V15	-	-	
MQTT broker	-	-	If you would like to encrypt the communication, the MQTT broker needs to support the SSL/TLS process.

This application example consists of the following components:

Table 3-4

Component	File name
"LMqtt” and "LMqttQdn" library	109748872_MqttClient_Publish_Secure_CODE_V1_1.zip
This document	109748872_MqttClient_Publish_Secure_DOKU_V1_1_en.pdf

Note To reach the broker with a static ip address, please use the lobrary "LMqtt".

To reach the broker with a "Qualified Domain Name", please use the "LMqttQdn".

2 Engineering

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

2.1 Block description

2.1.1 “LMqtt_Publisher” interface description

Note The function block is "L;qtt_Publsiher" is available in all libraries and identical.

The function block is designed for an “optimized block access”.

In the following charts, the input and output parameters of the “LMqtt_Publisher” function block are explained.

Input parameters

Table 2-1

Parameter	Data type	Function
enable	BOOL	The function block is activated with a positive edge. The function block is active as long as the state of “enable” is “true”. Via a negative edge, the function block is terminated and the TCP and MQTT connection is disconnected.
publish	BOOL	A message is sent to the broker with a positive edge.
tcpConnParam	"typeTcpConnParam"	Data area of TCP connection information
mqttParam	"typeMqttParam"	Data area of MQTT connection and message information

Output parameters

Table 2-2

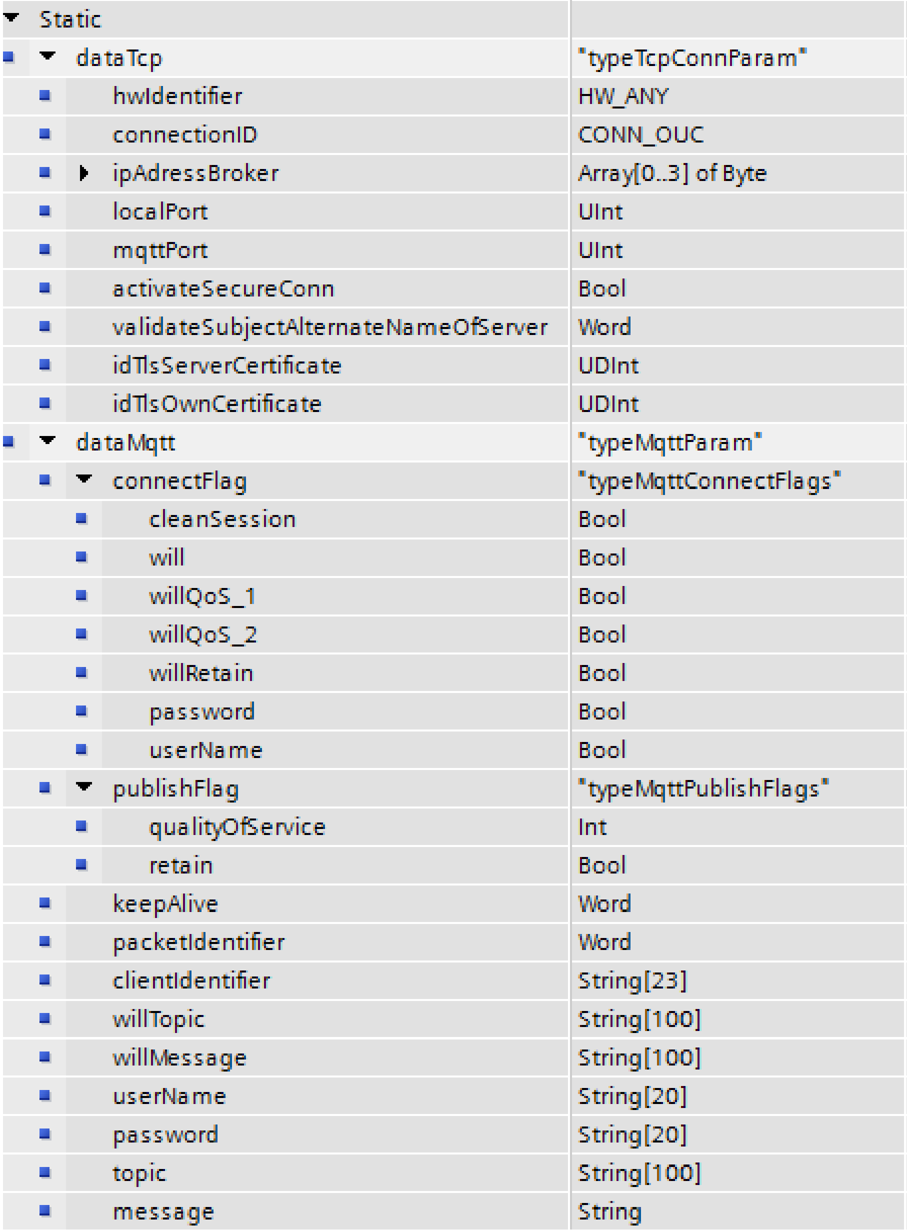
Parameter	Data type	Function
tcpConnected	BOOL	True, if connection is established
mqttConnected	BOOL	True, if the MQTT connection is established
published	BOOL	True, if the message successfully arrived at the broker. It is on “true” for one cycle only.
busy	BOOL	True, while a message or a PING is being sent to the broker
error	BOOL	True, if an error is present
statusID	INT	Status that triggered the error
status	DWORD	Error message

2.1.2 “LMqtt_Data” data block

Note The data block is designed for an “optimized block access”.

In the figure below, you can see the declaration of the data block:

Figure 2-1



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

Data types overview

To structure the data volume in a clear way, several data types have been created. In the list below, you can see the data types used in the program:

- "typeTcpConnParam"
- "typeMqttParam"; subdivided in
  - "typeMqttConnectFlags"
  - "typeMqttPublishFlags"

"typeTcpConnParam” data type

In this data type, all information required to establish the TCP connection are stored. You can set these parameters according to your specifications.

Table 2-3

Parameter	Data type	Meaning
hwIdentifier	HW_ANY	HW-ID of the PROFINET interface of the CPU
connectionId	CONN_OUC	ID of the TCP connection
ipAdressBroker	Array[0..3] of BYTE	IP address of broker, e.g. for the address 192.168.0.10 ipAdressBroker[0] is "192" ipAdressBroker[1] is "168" ipAdressBroker[2] is "0" ipAdressBroker[3] is "10"
localPort	UINT	Local port number in the CPU
mqttPort	UINT	Remote port at MQTT broker
activateSecureConn	BOOL	True, if the communication is to be secured via TLS
validateSubjectAlternateNameOfServer	WORD	A set bit 0 means that the TLC client validates the alternative name of the certificate holder. Bit 1 to 15 are reserved. Only relevant, if "activateSecureConn" is "true"
idTlsServerCertificate	UDINT	Identifier of X.509-V3 certificate (usually a CA certificate) to validate the authentication of the TLS server. If this parameter is “0”, the TLS client uses all (CA) certificates for the validation of the server authentication that are currently loaded in the certificate memory of the client. Only relevant, if "activateSecureConn" is "true"
idTlsClientCertificate	UDINT	ID of own X.509-V3 certificate to validate the own authentication against the TLS server. Only relevant, if "activateSecureConn" is "true” and the TLS server requests a client authentication.

Note In this application example, the MQTT broker (TLS server) waives the authentication of the TLS client (here: SIMATIC S7-1500). In this case, the “idTlsClientCertificate” parameter is unused.

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

"typeMqttParam" data type

This data type contains all information on MQTT. In the list below, you can see which information you can store here:

- Flags for connection establishment
- Flags for sending messages
- Login information at broker
- Topic
- Message text

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

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

Table 2-4

Parameter	Data type	Meaning
cleanSession	BOOL	True, if all data from a previous session are to be deleted.
will	BOOL	Activates the "last will and testament" function.
willQoS_1	BOOL	True, if the QoS has level 1 for the last will.
willQoS_2	BOOL	True, if the QoS has level 2 for the last will.
willRetain	BOOL	True, if the last will is to be saved as soon as it has been 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.

With the "typeMqttPublishFlags" data type, you can designate flags for the MQTT message.

Table 2-5

Parameter	Data type	Meaning
qualityOfService	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
retain	BOOL	True, if the message is to be saved at the broker.

The table below shows all further parameters of the “typeMqttParam” data type that you can designate for MQTT.

Table 2-6

Parameter	Data type	Meaning
keepAlive	WORD	Time interval of KeepAlive function in seconds. The time is indicated in hexadecimal notation. A KeepAlive with the value “0” deactivates the KeepAlive function. The maximum permissible time is 18h 12min 15 s.
packetIdentifier	WORD	Start value for package number. The number is automatically incremented in the program.
clientIdentifier	String [23]	Unique name of the client. With this name, the client identifies itself at the broker during the connection establishment. The following characters are permitted: • Figures • Lowercase and uppercase letters
willTopic	String [100]	If the will flag is set, the topic for the last will needs to be defined at this point.
willMessage	String [100]	If the will flag is set, the message for the last will needs to be defined at this point.
userName	String [20]	If the username flag is set, the user name for the login at the broker needs to be defined here.
password	String [20]	If the password flag is set, the password for the login at the broker needs to be defined here.
topic	String [100]	Name for the topic
message	String	Message text

Note

Please note the following rules:

1. If you set the “will” flag to “true”, you need to specify a character string at the “willMessage” and “willTopic” tags.
2. If you set the “will” flag to “false”, you also need to set the following flags to “false”:
   - "willQoS_1"
   - "willQoS_2"
   - "willRetain"
3. If you set the “username” and “password” flags to “true”, you need to specify a character string with the login data at the “userName” and “password” tags. These login data need to correspond to the login data you have stored at the MQTT broker.

2.2 Integration into the user project

Creating the TIA Portal project

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

Connect the SIMATIC controller and the MQTT broker via Ethernet.

Note

If you would like to use a secured MQTT communication via TLS, then at least firmware version 2.0 must be installed on the CPU.

Copying the blocks

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

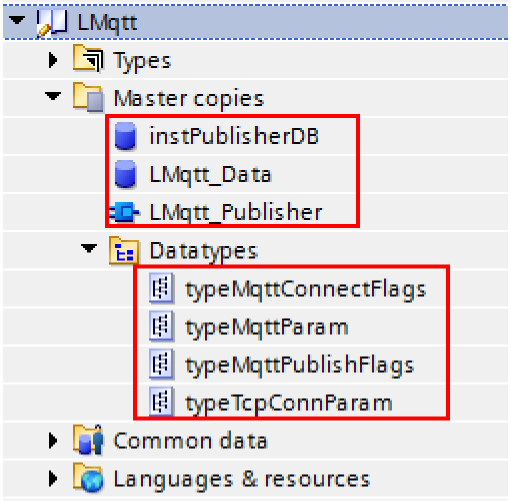
To copy the blocks to your TIA project, proceed as follows:

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

2. In TIA Portal, open the library view. In the toolbar of the "Global library" palette, click on the "Open global library" icon.
   The "Open global library" dialog opens.

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

4. Copy the content of "Master copies" into the respective folder of your project:
   - The function, data, and instance data block into your "Program files" folder
   - The data types into the "Data types" folder



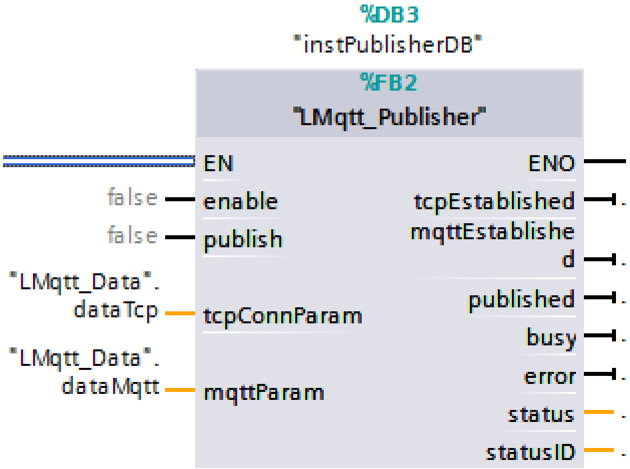
```
▼ LMqtt
  ▶ Types
  ▼ Master copies
      instPublisherDB
      LMqtt_Data
      LMqtt_Publisher
    ▼ Datatypes
        typeMqttConnectFlags
        typeMqttParam
        typeMqttPublishFlags
        typeTcpConnParam
  ▶ Common data
  ▶ Languages & resources
```

Calling and interconnecting the function block

After you have integrated the blocks into your project, you need to call up and interconnect the function block in your program.

1. Call up the “LMqtt_Publisher” function block, e.g. in OB 1 and assign an instance data block to it.

2. Interconnect the input or output tags to your preference. Only the interconnection of the input or output tags is dictated:
   - Input and output tag “tcpConnParam” with “LMqtt_Data”.dataTCP
   - Input and output tag “mqttParam” with “LMqtt_Data”.dataMqtt



2.3 Configuration of the security function

Note You only need to configure the security function, if you use a secured MQTT connection via TLS.

Note In this application example, the MQTT broker (TLS server) waives the authentication of the MQTT clients. Hence, only the CA certificate of the MQTT broker is required to authenticate the MQTT broker.
If you have configured the MQTT broker in such a way that an MQTT client authentication is also needed, then you have to import the client certificate as well.
The client certificate needs to be signed by the same CA as the server certificate.

The encryption through SSL/TLS works via certificates. A certificate is a public key signed by the holder. It guarantees the holder’s authenticity and integrity. For the authentication of the broker, the MQTT client requires the CA certificate of the broker.

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

Precondition for a TLS/SSL encryption

To establish a secured MQTT communication between the SIMATIC S7-CPU (MQTT client) and an MQTT broker in your network, the following criteria must be fulfilled:

- The MQTT broker is installed and preconfigured for the TLS process
- The required CA certificate of the MQTT broker is at hand
- The CPU's time of day has been set to the current time.
  A certificate always contains a validity period during which the certificate is valid. To be able to encrypt via the certificate, the S7-CPU’s time of day also needs to be within this validity period. On a brand-new S7-CPU or after fully resetting the S7-CPU, the internal clock is set to a default value that is outside the validity period of the certificate. The certificate is then marked as invalid.

2.3.1 Using the global certificate manager in TIA Portal

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

In TIA Portal, 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, you can, for example, import new certificates as well as export, update or replace already existing certificates. Every certificate is assigned to an ID via which the certificate can be referenced in the program blocks.

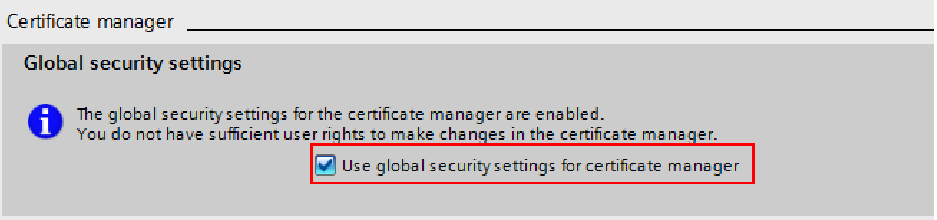
Enabling the global certificate manager

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

To be able to import and use the CA certificate of the MQTT broker, you need to activate the global certificate manager.

1. In the device or network view, select the CPU. The CPU properties are shown in the inspector window.

2. In the area navigation of the “Properties” tab, select the entry “Protection & Security > Certificate manager”. Tick the “Use global security settings for certificate manager” check box.



Certificate manager

Global security settings

The global security settings for the certificate manager are enabled.
You do not have sufficient user rights to make changes in the certificate manager.

☑ Use global security settings for certificate manager

Result

In the project navigation, the new entry “Security settings” appears.

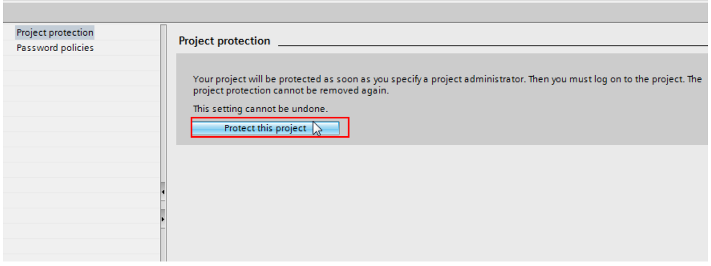
User Login

After having activated the global security settings for the certificate manager, you need to log in to the global security settings. Without a login, you will not be able to access the global certificate manager.

Log in to the global security settings as security user by proceeding as follows:

1. In the project navigation at “Security settings”, double-click the entry “Settings”.

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



Project protection

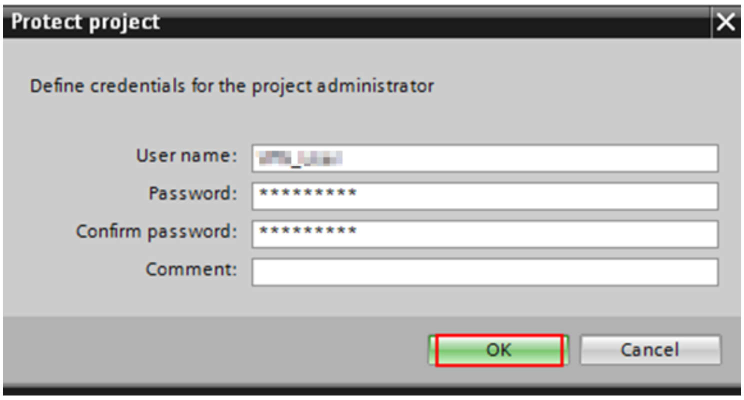
Your project will be protected as soon as you specify a project administrator. Then you must log on to the project. The project protection cannot be removed again.

This setting cannot be undone.

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".



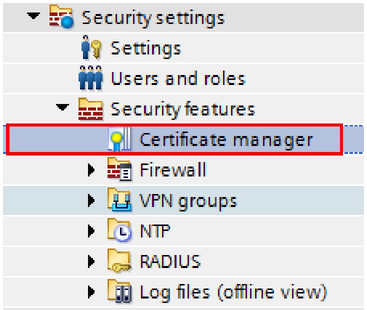
Protect project

Define credentials for the project administrator

User name: [redacted]
Password: *********
Confirm password: *********
Comment:

Result

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

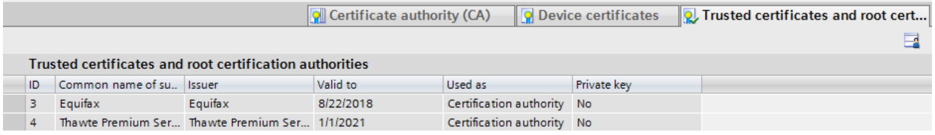


Using the global certificate manager

With the global certificate manager, you now have the possibility to import external certificates into TIA Portal. By double-clicking the “Certificate manager” line, you gain access to all certificates in the project. They are divided into the tabs “CA” (certification authority), “Device certificates” and “Trusted certificates and root certification authorities”.

1. In the project navigation at “Global security settings”, double-click the entry “Certificate manager”.

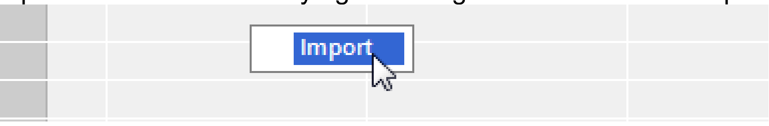
2. Select the appropriate table for the certificate that is to be imported (CA certificate, device certificate, trusted certificates of root certification authorities).


Certificate authority (CA)
Device certificates
Trusted certificates and root cert...

Trusted certificates and root certification authorities

ID	Common name of su...	Issuer	Valid to	Used as	Private key
3	Equifax	Equifax	8/22/2018	Certification authority	No
4	Thawte Premium Ser...	Thawte Premium Ser...	1/1/2021	Certification authority	No

3. Open the context menu by right clicking in the table. Click “Import”.

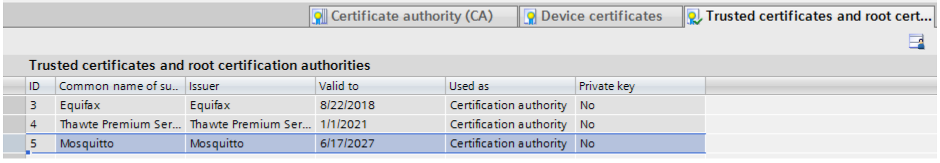


4. Select the import format of the certificate:
   - CER, DER, CRT or PEM for certificates without private key
   - P12 (PKCS12 archive) for certificates with private key.

   To import the certificate, click on “Open”.

Result

The CA certificate of the MQTT broker is now in the global certificate manager. In the next chapter, you manually assign the CA certificate to the CPU.


Certificate authority (CA)
Device certificates
Trusted certificates and root cert...

Trusted certificates and root certification authorities

ID	Common name of su...	Issuer	Valid to	Used as	Private key
3	Equifax	Equifax	8/22/2018	Certification authority	No
4	Thawte Premium Ser...	Thawte Premium Ser...	1/1/2021	Certification authority	No
5	Mosquitto	Mosquitto	6/17/2027	Certification authority	No

Note

If the MQTT broker additionally requires an authentication of the MQTT client, you need to import the client certificate.
Please note the following:

- The client certificate needs to be signed by the same CA as the server certificate.
- The global certificate must be imported into the global certificate manager as .pk12 container (with certificate and private key)
- The client certificate must be imported into the “Device certificates” table.

2.3.2 Using the local certificate manager of the CPU

For the time being, the CA certificate is located only in the global certificate manager of TIA Portal. Certificates that have been imported into the global security settings via the certificate manager are not automatically assigned to the respective modules.

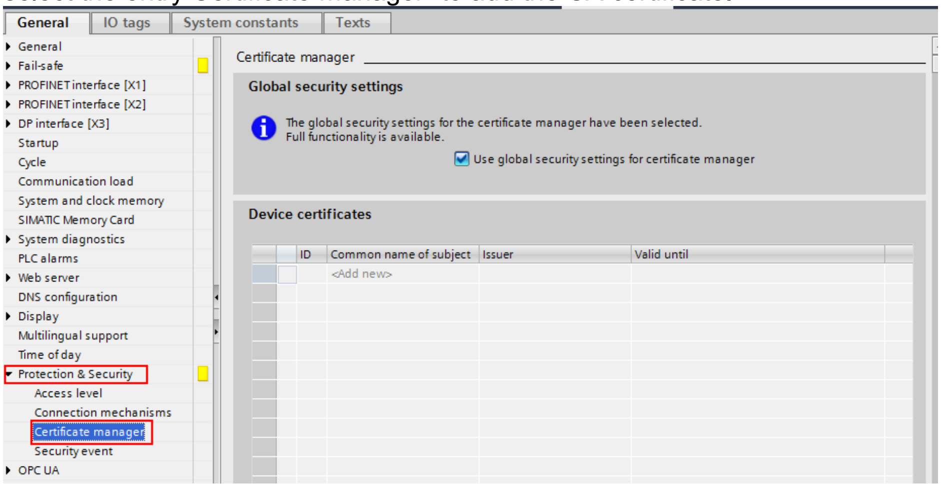
In order to authenticate the MQTT broker, you have to load the CA certificate into the CPU. Only those device certificates are loaded to the module, which you have assigned as device certificates in the module via the local certificate manager.

This assignment is done 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 during 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 CPU properties are shown in the inspector window.

2. In the area navigation of the “Properties” tab under “Protection & Security”, select the entry Certificate manager” to add the CA certificate.



Certificate manager

Global security settings

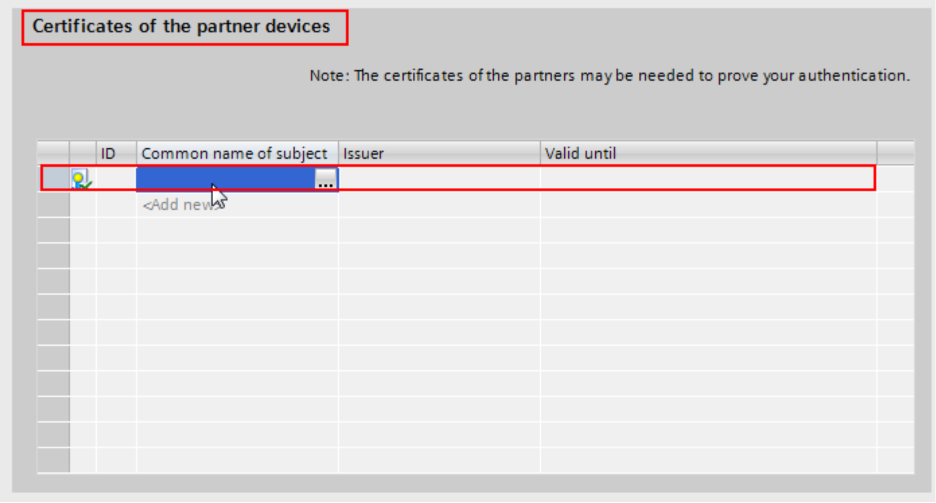
The global security settings for the certificate manager have been selected.
Full functionality is available.

☑ Use global security settings for certificate manager

Device certificates

ID	Common name of subject	Issuer	Valid until
	<Add new>		

3. Go to the section “Certificates of the partner devices”. In the table of the certificates, click on “Add”. A new line is added to the table.

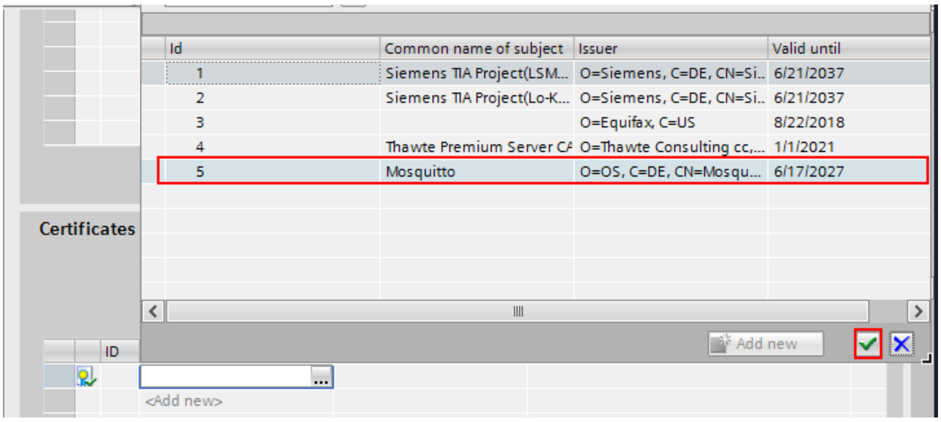


Certificates of the partner devices

Note: The certificates of the partners may be needed to prove your authentication.

ID	Common name of subject	Issuer	Valid until
	...		
	<Add new>		

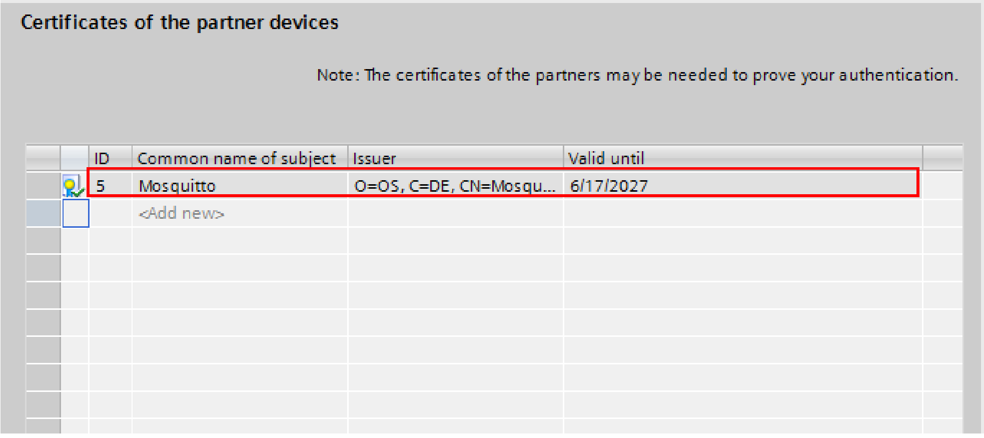
4. Click into the new line. The selection for new certificates opens. Select the previously imported CA certificate from the global certificate manager. Click on the green check mark to add the selection to the table of the certificates.



Id	Common name of subject	Issuer	Valid until
1	Siemens TIA Project(LSM...	O=Siemens, C=DE, CN=Si...	6/21/2037
2	Siemens TIA Project(Lo-K...	O=Siemens, C=DE, CN=Si...	6/21/2037
3		O=Equifax, C=US	8/22/2018
4	Thawte Premium Server CA	O=Thawte Consulting cc,...	1/1/2021
5	Mosquitto	O=OS, C=DE, CN=Mosqu...	6/17/2027

Result

The selected certificate has been assigned to the CPU and has been given an ID. The ID is the number of the certificate. Enter this value into the connection parameters for the “idTlsServerCertificate” parameter (see chapter 2.1.2).



Certificates of the partner devices

Note: The certificates of the partners may be needed to prove your authentication.

ID	Common name of subject	Issuer	Valid until
5	Mosquitto	O=OS, C=DE, CN=Mosqu...	6/17/2027
	<Add new>		

Note If the MQTT broker additionally requires an authentication of the MQTT client, you also need to assign the imported client certificate to the CPU (section “Device certificates”). Enter this value of the ID into the connection parameters for the “idTlsClientCertificate” parameter (see chapter 2.1.2).

2.4 Parameterization and operation

Setting the parameters

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

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

You have to enter your own value for the following parameters in particular:

- IPV4 address or domain name of the MQTT broker. The domain name must ends with a ".".
- Remote port at which the MQTT broker receives the messages
- Parameters for the secured communication
  - Status of the security function (On/Off) for this connection
  - ID of CA certificate (only relevant for secured connections)
  - ID of own certificate, in case the MQTT broker also authenticates the client (only relevant for secured connections)
- All MQTT parameters, for example
  - Flags for connection establishment
  - Flags for sending messages
  - Login information at broker
  - Topic
  - Message text

Download the project into your CPU.

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

Operating the application example

If all parameters are set and the CA certificate of the MQTT broker is added to the local certificate manager of the CPU, you can test the application example.

Prior to testing the application example, check the following points:

1. The project has been loaded to the CPU.
2. The CPU and the MQTT broker are connected to each other via Ethernet and accessible.
3. The MQTT broker is adequately configured and launched.
4. The login on the MQTT broker is launched in order to additionally support the login of the MQTT clients and the publish mechanism.

If the above mentioned criteria are met, you may initiate the MQTT communication between the CPU and the MQTT broker. For this, trigger the “enable” tag at the “LMqtt_Publisher” function block by means of a positive edge.

In a positive case, the internal state machines are passed through. They then establish a TCP or MQTT connection to the MQTT broker. The output tags “tcpConnected” and “mqttConnected” are set and indicate an established TCP or MQTT connection.

You are now able to send an MQTT message. For this, trigger the “publish” input tag.

If the connection to the MQTT broker is not being established, check the “status” and “statusID” output tags to diagnose the error. The meaning of the values of both tags can be found in chapter 2.5.

2.5 Error handling

If an error occurs in the program, the current status of the state machines and the cause of error are written into the “statusID” and “status” output tags.

“statusID”

At the “statusID” output, the number of the status is output in which the error has occurred. The states are numbered consecutively and have the following meaning.

Table 2-7

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

"status"

The “status” output parameter displays the error code:

Table 2-8

statusID	status	Description	Remedy
-1	Status message from “TDISCON” block	See manual	-
2	Status message from “TCON” block	See manual	Check the accessibility of the broker. IP address, port, firewall
3	Status message from “TRCV” block	See manual	Check the network connection
10	W#16#80F0	Error at “Will” flag	Check the flags in the “typeMqttConnectFlags” data type
	W#16#80F1	Error at “WillQoS” flag	
	W#16#80F3	Error at “KeepAlive” flag	KeepAlive must be greater than 2 seconds.
11	Status message from “TSEND” block	See manual	-
12	1	The broker does not accept the MQTT protocol level	Check the access data in the “typeMqttParam” data type
	2	ClientIdentifier is not accepted	
	3	MQTT service not existent	
	4	Data in username/password are incorrect	
	5	Client is not authorized	
13	Status message from “TSEND” block	See manual	-
14,19,20	W#16#80F2	Wrong packetIdentifier received	-
20	Status message from “TRCV” block	Timeout	Check the connection parameters

3 Valuable information

3.1 Basics of MQTT

Note A detailed description on MQTT can be found in the MQTT specification description (see \3\ in chapter 4.2).

3.1.1 Terminology

The most important terms for the telemetry protocol MQTT are described below.

MQTT message

A message in MQTT consists of several parts:

- A defined topic
- An assigned characteristic for the Quality of Service.
- 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 execute the following functions:

- Sending messages with a defined topic to the broker which could be of interest to other clients (publish mechanism)
- Subscribing to messages at the broker which follow a specified topic (subscriber mechanism)
- Logging itself out from subscribed messages
- Terminating the connection to the broker

Note The “LMqtt_Publisher” function block in this application example supports the following functions:

- Publish mechanism
- Logout from broker.

MQTT broker

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

- Accepting network connections from the clients
- Receiving messages from an MQTT client
- Processing subscription requests from MQTT clients
- Forwarding messages to the MQTT clients that match their subscriptions

Note The MQTT broker is not part of this application example and is assumed.

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). It is the task of the sender of a message (publisher mechanism) to define the contents and the topic of a message to be sent. It is then the broker’s task to ensure that the subscribers receive the message from the subscribed topics. The topics follow a defined pattern. They are similar to a directory path and form 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 protocol used, MQTT runs on different access ports. Offered ports are:

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

Architecture

MQTT is a publish and subscribe protocol. This mechanism disconnects a client that sends messages (publisher) from one or more clients which receive the messages (subscriber). This also means that the “publishers” do not know 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 takes care of communication control.

3.1.3 Features

MQTT offers useful features.

Quality of Service

For the Quality of Service during messaging, the MQTT specification provides three levels:

- QoS "0": The lowest level 0 involves a "fire'n'forget" process. There is therefore no guarantee that the message will be received at all.
- QoS "1": At QoS level 1, it is ensured that the message will reach the topic queue at least once. The broker acknowledges the receipt of the message.
- QoS "2": At the highest level 2, the broker guarantees via several handshakes with the client, that the message will be stored exactly once.

Last will

MQTT supports the "last will and testament" function. This function is used to notify other clients, if the connection to a client has been unexpectedly terminated.

During the connection establishment to the broker, every client can specify its last will and inform the broker on the same. This last will is structured like an ordinary MQTT message, including topic, QoS and payload. The broker saves the last will. As soon as the broker notices that the connection to the respective client has been unexpectedly terminated, it sends the last will as MQTT message to all subscribers that have registered for the topic. This way, the subscribers are also informed about the disconnected client.

KeepAlive

MQTT supports the KeepAlive function. The KeepAlive function ensures that the connection remains open as soon as client and broker are connected to each other.

For the KeepAlive, the clients define a time interval and inform the broker about it during their connection establishment. This interval is the maximum tolerated time period in which the client and the broker are allowed to remain without any contact. If this time is exceeded, the broker must disconnect.

This means that, as long as the client regularly sends messages to the broker within the KeepAlive interval, the client must not perform any particular action in order to maintain the connection. However, if the client does not send any messages within the KeepAlive interval, it must send a ping package to the broker before the time period has elapsed. With this PING, the client signals to the broker that it continues to be accessible.

If a message or ping package has been sent to the broker, the timer for the KeepAlive interval starts all over again.

Note

- The client determines the KeepAlive interval. This way, it can adjust the interval to its environment, e.g. due to a low bandwidth.
- The maximum value for the KeepAlive interval is 18 h 12 m 15 s
- If the client sets the KeepAlive interval to the value "0", the KeepAlive mechanism will be deactivated.

Message persistence

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

Retained messages

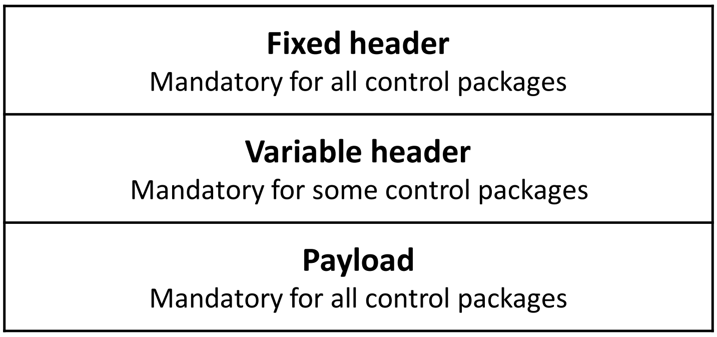
If an MQTT client subscribes to a topic for the first time, it usually receives a message only when another MQTT client sends a message with the subscribed topic the next time. With “Retained messages”, the subscriber immediately receives the last value that has been sent to the topic, prior to its subscription request.

3.1.4 MQTT control packages

Most MQTT control packages operate on the principle of the handshake process. The MQTT client is always the active element and transmits a task to the broker. Depending on the task, the broker confirms the request.

The structure of an MQTT control package is fixed. The following graphic shows the structure:

Figure 3-1



Fixed header
Mandatory for all control packages

Variable header
Mandatory for some control packages

Payload
Mandatory for all control packages

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

- An ID number for the MQTT control package type
- An area for possible flags; if there are no flags intended for the control package, the bits are marked as "reserved".
- The number of the following bytes after the "Fixed Header"

The "Variable Header" is only required for some control packages. The content of the "Variable Header" depends on the control package type.

The payload is mandatory for most control packages. Here, the content also depends on the control package type. For every control package type, there are clear rules about with what and in which sequence the payload may be filled.

Note A detailed description on the MQTT control packages can be found in the MQTT specification description (see \3\ in chapter 4.2).

The MQTT control packages from this application example are briefly described in the following sections.

MQTT connection

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

The connection is initiated by a client as soon as the client sends a "CONNECT" package to the broker. In the positive case, the broker answers with a "CONNACK" package and a status code.

In the following cases, the broker immediately shuts down the connection:

- If the "CONNECT" package is damaged
- If the structure of the "CONNECT" package does not match the specifications
- If the connection establishment takes too long

In the “Variable Header”, a “CONNECT” package contains an area for flags. The “CONNECT” flag byte contains a set of parameters that specify the behavior of the MQTT connection. The “CONNECT” flag byte also shows which optional fields do or do not exist in the “payload”.

In the “payload”, the following fields are mandatory:

- The “ClientID” serves to identify the client at the broker
- Via the “CleanSession”, the connection type can be regulated
- Via the KeepAlive time, the time interval is determined during which the client needs to respond to the broker. This can be done either via sending a message or a PING command. If the client does not respond in the given time interval, the broker disconnects from the client.

Optional fields are, for example, username, password and information on the “last will”.

MQTT push mechanism

As soon as an MQTT client has connected to a broker, it can send messages to the broker. For this, the client uses the “PUBLISH” package. As messages in MQTT are filtered and managed topic-based, every MQTT message needs to contain a topic. The topic is part of the “Variable Header”. The actual message text is included in the “payload”.

Depending on the settings of the Quality of Service (“QoS”), the push mechanism either ends at this point, or further control packages are exchanged:

If QoS is “0”, the send request ends at this point.

If QoS is “1”, the broker acknowledges the “PUBLISH” package with a “PUBACK”.

If QoS is “2”, the broker acknowledges the “PUBLISH” package with a “PUBREC”. After this, another handshake between client and broker is done. The client answers the “PUBREC” with a “PUBREL” package. The broker completes the double handshake with a “PUBCOM” package.

Note Detailed information on the Quality of Service QoS can be found in chapter 3.1.3.

MQTT ping mechanism

If the KeepAlive function is active (the KeepAlive interval is greater than “0”), the client needs to send at least one message to the broker within the KeepAlive interval. If this is not the case, the broker needs to disconnect from the client. In order to avoid this kind of forced disconnection, the client needs to send a ping request to the broker before the KeepAlive time has elapsed. For this, the “PINGREQ” control package is used. The broker answers with a “PINGRESP” package and thus signals its accessibility to the client.

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

MQTT disconnection

A client can disconnect from a broker by sending a “DISCONNECT” package to the broker. The broker then deletes all “last will and testament” information. Since the client is active and connected out of its own free will, the broker does not send its last will to the registered subscribers.

3.2 Details on the mode of operation of the "LMqtt_Publisher" FB

3.2.1 Requirements and information

For a communication between an MQTT client and an MQTT broker, the following requirements must be fulfilled:

1. A TCP connection to the MQTT broker has been successfully established (Status: "TCP_CONNECTED").
2. Via the established TCP connection, the "LMqtt_Publisher" function block has logged into the broker as MQTT client and connected to it (Status: "MQTT_CONNECTED").
3. The trigger for sending the message or for maintaining the MQTT connection ("KeepAlive") is active. Depending on the desired Quality of Service, the message will be send to the broker via the established MQTT connection.

Note

An MQTT connection establishment is only possible, if the TCP connection to the broker is successfully established and will be maintained.

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

Overview

In order to fulfill the mentioned requirements, several state machines have been implemented in the program:

- State machine "TCP": Managing the TCP connection
- State machine "MQTT": Managing the MQTT connection
- State machine "PUSH": Processing the send procedure

3.2.2 State machine "TCP"

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

- It manages the TCP connection establishment
- It monitors the established TCP connection for connection errors, e.g. cable break
- If an error has occurred or if no positive edge has been detected at the "enable" input parameter, it sets all static tags and all other state machines into a defined state.

The "TCP" state machine contains the following states:

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

The following table shows the meaning of the states

Table 3-1

State	Description
IDLE	In the idle state “IDLE”, all parameters are reset. The state machine remains in this state until it has detected a positive edge at the “enable” input parameter. As soon as there is a positive edge at the input, the state machine will be put into the “TCP_PARAM” state.
TCP_PARAM	In this state, all connection parameters are read in. Without step enabling condition, the function block switches to the “TCP_CONNECTING” state.
TCP_CONNECTING	In this state, the TCP connection to the MQTT broker is established. If the connection has been successfully established with “TCON”, the FB switches to the “TCP_CONNECTED” state and the “tcpConnected” output tag is set. The TCP connection will be maintained until it is terminated with “TDISCON”. If an error occurs during the connection establishment, the state machine switches to the “TCP_ERROR” state.
TCP_CONNECTED	The function block remains in this state until the following events occur: • The “TRCV” block recognizes a disconnection, e.g. by pulling the network cable, and signals an error. • The “enable” input parameter is reset and thus initiates the disconnection. If the “TRCV” block detects an error, the state machine switches to the “TCP_ERROR” state. The “TCP_CONNECTED” state is a prerequisite for the processing of the “MQTT” state machine.
TCP_DISCONNECT	In this state, the TCP connection is terminated. If the “TDISCON” block detects an error, the state machine switches to the “TCP_ERROR” state.
TCP_ERROR	If an error occurs in the “TCP” state machine, the “TCP_ERROR” state is the central contact point. Here, the required parameters (static tags and output tags) are set or reset and the MQTT connection is terminated. The following actions are also performed: • The error message of the participating T block is returned at the “status” output. • At the “statusID” output, the number of the status is output in which the error has occurred. • The state machine returns to the “IDLE” state. If there is already an established TCP connection, it will first be terminated. The “tcpConnected” output tag is reset. • The “MQTT” state machine is put into the MQTT_DISCONNECTED” state.

Note In the event of an error, the “LMqtt_Publisher” function block is not “self-healing”. This means that the function block falls back into the “IDLE” state and remains there until it has once again detected a positive edge at the “enable” input tag.

3.2.3 State machine “MQTT”

The “MQTT” state machine is started automatically, if the “TCP” state machine reaches the “TCP_CONNECTED” state. This state machine has the following functions:

- It manages the handshake procedure to establish the MQTT connection
- It handles the disconnection
- It manages the internal “PUSH” state machine to send messages
- It takes care of a PING package being sent before the KeepAlive interval elapses.

The “MQTT” state machine contains the following states

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

The following table shows the meaning of the states:

Table 3-2

State	Description
MQTT_DISCONNECTED	As long as no TCP connection is established, the state is always “MQTT_DISCONNECTED”. Only when a TCP connection has been established, is the step enabling condition is automatically activated to the “MQTT_CONNECT_FLAG_CHECK” state.
MQTT_CONNECT_FLAG_CHECK	In this state, the flags and parameters for the MQTT connection establishment are read in and validated. If discrepancies occur during the verification, the state machine will switch to the “MQTT_ERROR” state and a corresponding error message will be output at the “status” output parameter. In an error-free state, the state machine switches to the “MQTT_CONNECT” state, without step enabling condition.
MQTT_CONNECT	In this state, the MQTT connection to the MQTT broker is established. For this, a “CONNECT” package with the read-in parameters is created and send to the broker via the “TSEND” block. If an error occurs while sending the “CONNECT” package, the state machine switches to the “MQTT_ERROR” state. If the “CONNECT” package has been successfully sent, the state machine switches to the “MQTT_CONNACK” state.
MQTT_CONNACK	The state machine remains in this state until the “TRCV”block receives a message. It is checked whether it is a “CONNACK” package. When the broker has confirmed the connection request with “CONNACK”, the state machine switches to the “MQTT_CONNECTED” state and the “mqttConnected” output tag is set. The KeepAlive interval is started, if needed. If the “TRCV” block detects an error, the state machine switches to the “MQTT_ERROR” state.
MQTT_CONNECTED	The function block remains in this state until the MQTT connection or TCP connection is terminated. In the “MQTT_CONNECTED” state, the following points are cyclically checked: • Is there a transmission instigation for an MQTT message? • Is the KeepAlive interval about to end and a PING

State	Description
	command needs to be send to the broker? Depending on the check result, the internal “PUSH” state machine is put into the corresponding state in order to execute the desired routine.
MQTT_DISCONNECT	If the “enable” input tag is reset, the MQTT connection will be terminated. For this, a “DISCONNECT” package is created and send to the broker via the “TSEND” block. If an error occurs while sending the “DISCONNECT” package, the state machine switches to the “MQTT_ERROR” state. If the “DISCONNECT” package has been successfully sent, the state machine switches to the “MQTT_DISCONNECTED” state. At the same time, the “TCP” state machine is put into the “TCP_DISCONNECT” state. With this, the TCP connection will be terminated as well.
MQTT_ERROR	If an error occurs in the “MQTT” state machine, the “MQTT_ERROR” state is the central contact point. Here, the required parameters (static tags and output tags) are set or reset. The following actions are also performed: • The error message of the participating MQTT command is returned at the “status” output. • At the “statusID” output, the number of the status is output in which the error has occurred. • The state machine returns to the “MQTT_DISCONNECTED” state.

3.2.4 State machine “PUSH”

The “PUSH” state machine is passed through only if the “MQTT” state machine is in the “MQTT_CONNECTED” state. This is because here, it is decided from which point the “PUSH” state machine will be started. If there is a transmission instigation for an MQTT message, the sending routine becomes active. If the KeepAlive time is about to end, the PING routine is started.

The “PUSH” state machine contains the following states:

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

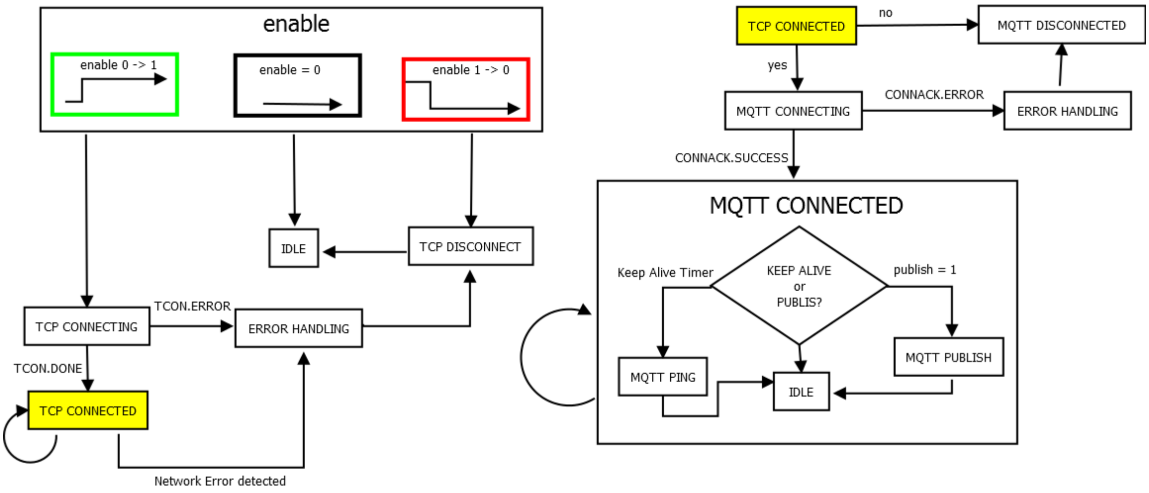
State	Description
IDLE	As long as there is no transmission instigation or the KeepAlive interval is not about to elapse, the state is always “IDLE”.
MQTT_PUBLISH	If in the “MQTT_CONNECTED” state at the “publish” input tag, a positive edge has been detected, the internal “PUSH” state machine will be put into the “MQTT_PUBLISH” state. Here, the sending routine starts, depending on the Quality of Service (QoS). First, a “PUBLISH” package with the specified parameters, the topic and the message text will be put together and then sent to the broker via the “TSEND” block. If an error occurs while sending the “PUBLISH” package, the “MQTT” state machine switches to the “MQTT_ERROR” state and the state machine returns to “IDLE”. If the “PUBLISH” package has been successfully sent, then the next step depends on the selected QoS: • If QoS is “0”, the sending process ends at this point and this state machine returns to “IDLE”. The KeepAlive interval is restarted, if needed. • If QoS is “1” and “2”, this state machine switches to the “MQTT_PUBACK” state in order to receive an acknowledgment from the broker.
MQTT_PUBACK	If the QoS is greater than “0”, the client awaits an acknowledgment of the “PUBLISH” package from the broker. The state machine remains in this state until the “TRCV” block receives a message. It is checked whether it is a “PUBACK” package. If the broker has acknowledged the receipt of the message, then the next step depends on the selected QoS: • If QoS is “1”, the sending process ends at this point and this state machine returns to “IDLE”. The KeepAlive interval is restarted, if needed. • If QoS is “2”, this state machine switches to the “MQTT_PUBREL” state in order to confirm the acknowledgment receipt. If the “TRCV” block detects an error, the “MQTT” state machine switches to the “MQTT_ERROR” state and the state machine returns to “IDLE”.
MQTT_PUBREL	If QoS is “2”, a double handshake with the broker is performed.

State	Description
	After the client has received the “PUBACK”, it is confirmed by the “PUBREL” package. For this, a “PUBREL” package is put together and then sent to the broker via the “TSEND” block. If an error occurs while sending the “PUBREL” package, the “MQTT” state machine switches to the “MQTT_ERROR” state and the state machine returns to “IDLE”. If the “PUBREL” package has been successfully sent, the state machine switches to the “PUBCOMP” state.
MQTT_PUBCOMP	This state is the last part of the double handshake procedure, if QoS is “2”. The client awaits an acknowledgment of the “PUBREL” package from the broker. The state machine remains in this state until the “TRCV” block receives a message. It is checked whether it is a “PUBCOMP” package. If the broker has acknowledged the receipt of the message, this state machine returns to “IDLE” and the KeepAlive interval is restarted, if needed. The handshake procedure is thus completed. If the “TRCV” block detects an error, the “MQTT” state machine switches to the “MQTT_ERROR” state and this state machine returns to “IDLE”.
MQTT_PING	If in the “MQTT_CONNECTED” state, it is found that the KeepAlive interval elapses, the internal state machine is put into the “MQTT_PING” state. Here, the ping routine starts. First, a “PING” package is put together and then sent to the broker via the “TSEND” block. If an error occurs while sending the “PING” package, the “MQTT” state machine switches to the “MQTT_ERROR” state and the state machine returns to “IDLE”.
MQTT_PINGRESP	After the PING package, the client awaits an acknowledgment from the broker. The state machine remains in this state until the “TRCV” block receives a message. It is checked whether it is a “PINGRESP” package. If the broker has acknowledged the receipt of the message, the state machine returns to “IDLE”. The KeepAlive interval is restarted. If the “TRCV” block detects an error, the “MQTT” state machine switches to the “MQTT_ERROR” state and this state machine returns to “IDLE”.

3.2.5 Function chart

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

Figure 3-2



4 Annex

4.1 Service and support

Industry Online Support

Do you have any questions or need assistance?

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

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

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

Technical Support

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

Service offer

Our range of services includes the following:

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

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

Industry Online Support app

Thanks to the "Siemens Industry Online Support" app, you will get optimum support even when you are on the move. The app is available for Apple iOS, Android and Windows Phone.
https://support.industry.siemens.com/cs/ww/en/sc/2067

4.2 Links and literature

Table 1-2

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 3-4

Version	Date	Modifications
V1.0	07/2017	First version
V1.1	08/2018	Add Library "LMqttQdn"