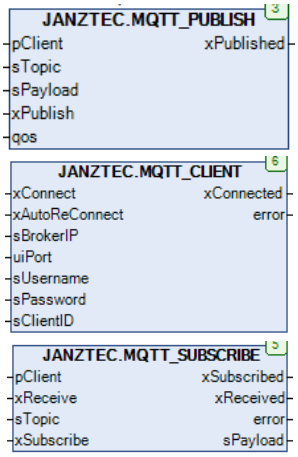
Data Sheet Janz Tec MQTT library for CODESYS

Product Description

MQTT is a Client Server publish/subscribe messaging transport protocol. It is light weight, open, simple, and designed so as to be easy to implement. These characteristics make it ideal for use in many situations, including constrained environments such as communication in Machine to Machine (M2M) and Internet of Things (IoT) contexts where a small code footprint is required and / or network bandwidth is limited.

Janz Tec developed a CODESYS library which implements the MQTT protocol. This way users can add the MQTT protocol easily to their own IEC 61131 applications. The Janz Tec MQTT library for CODESYS implements client functionality. It is completely written in IEC code which guarantees the library can be used on any type of target system.

Range of Functions



Function Blocks

Features

- CODESYS client library implementing MQTT protocol (version 3.1.1)
- MQTT client functionality
- Library completely written in IEC code, therefore independent from target system
- Publish: Payload of any type can be transmitted to a broker*
- Subscribe: Messages of any type from MQTT brokers can be received*
- Message classification via MQTT topics
- MQTT-QoS-Levels:
    - QoS0 (At-most-once)
    - QoS1 (At-least-once)
- Authentication with user name / password
- Auto-Reconnect is available if connection has been aborted
- Channel encryption via SSL (TLS v1.1 Client)
- Last will functionality
- Topic wildcards (+/#)

* For maximum supported message payload size, see global variable MQTT_GVL.MAX_PAYLOAD_SIZE

Currently not supported

- QoS2 (Exactly once)

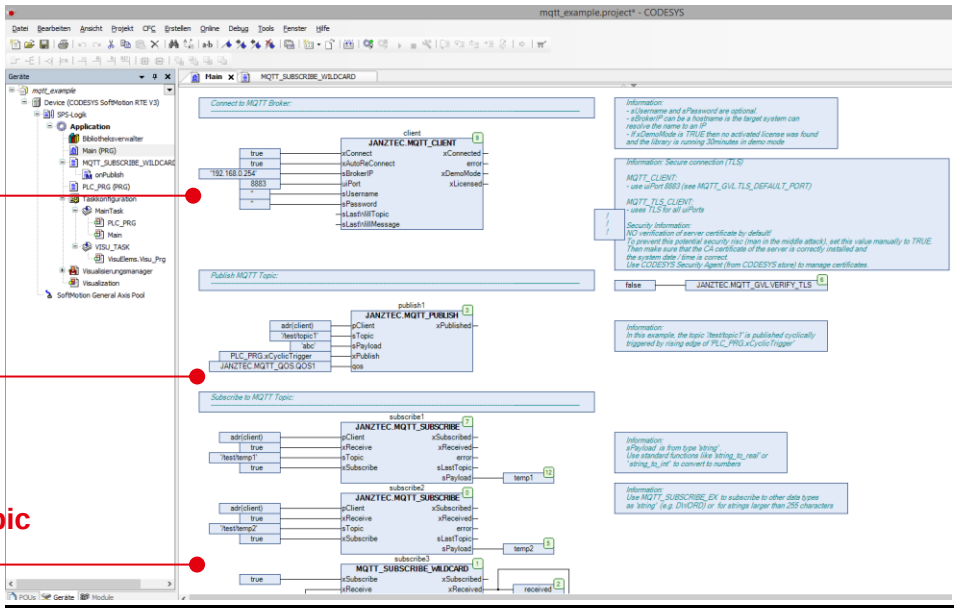
Product Screenshots

Sample Project: „mqtt_example.project“

Connect to MQTT Broker

Publish MQTT Topic

Subscribe to MQTT Topic



Usage Examples

Step 1:

Configure the IP address and port number of your MQTT broker in the example project „mqtt_example“ and start the application.

Step 2:

To update the gauge in the example project visualization, use an external system to publish a message with topic „/test/temp4“ and payload e.g. „12.00“ to your broker.

Information:

If you do not have a MQTT broker running in your network, consider using a Linux based computer as a MQTT broker, like e.g. a Raspberry Pi with Raspbian operating system. With the following commands you can easily install and start the Mosquitto MQTT broker:

root@raspberrypi:/ apt-get install mosquitto mosquitto-clients

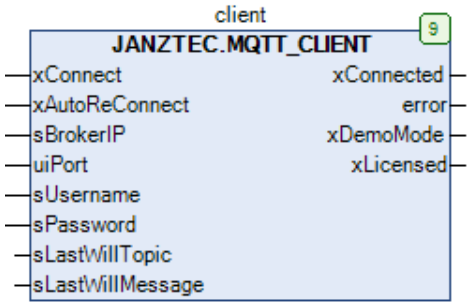
root@raspberrypi:/ service mosquitto start

root@raspberrypi:/ mosquitto_sub -h localhost -t "/test/+" -d

root@raspberrypi:/ mosquitto_pub -h localhost -t "/test/temp4" -m "12.00"

Reference

MQTT_CLIENT (FB)



FUNCTION_BLOCK MQTT_CLIENT

Client connects to MQTT broker (not encrypted).

If you need encryption, see MQTT_TLS_CLIENT

Scope	Name	Type	Initial	Comment
Input	xConnect	BOOL		HIGH: connect to sBrokerIP and uiPort. LOW: idle, or disconnect existing connection
	xAutoReConnect	BOOL	TRUE	HIGH: try every 1 second to reconnect after failure
	sBrokerIP	STRING	'127.0.0.1'	Broker IP address (or hostname) as string, eg: '192.168.0.1'
	uiPort	UINT	1883	Broker port, eg: 1883 (if port 8883 is specified, a TLS connection will be established. see MQTT_GVL.TLS_DEFAULT_PORT)
	sUsername	STRING	''	optional username, use empty string '' when not used
	sPassword	STRING	''	optional password, use empty string '' when not used
	sLastWillTopic	STRING	''	optional last will topic, use empty string '' when not used
	sLastWillMessage	STRING	''	optional last will message body, use empty string '' when not used
	xTLS	BOOL	FALSE	HIGH: encrypted connection using SSL/TLS. LOW: connect unencrypted
Output	xConnected	BOOL		HIGH: connection ready, LOW: not connected
	xDemoMode	BOOL		HIGH: if running in 30 minutes demo mode (no license found)
	xLicensed	BOOL		HIGH: if license is valid, LOW: no valid license found
	error	*MQTT_ERROR*		see MQTT_ERROR

MQTT_TLS_CLIENT (FB)

FUNCTION_BLOCK PUBLIC MQTT_TLS_CLIENT EXTENDS *MQTT_CLIENT*

Client connects to MQTT broker over secure connection (SSL/TLS)

Scope	Name	Type	Initial	Comment	Inherited from
Input	xConnect	BOOL		HIGH: connect to sBrokerIP and uiPort. LOW: idle, or disconnect existing connection	*MQTT_CLIENT*
	xAutoReConnect	BOOL	TRUE	HIGH: try every 1 second to reconnect after failure	*MQTT_CLIENT*
	sBrokerIP	STRING	'127.0.0.1'	Broker IP address (or hostname) as string, eg: '192.168.0.1'	*MQTT_CLIENT*
	uiPort	UINT	1883	Broker port, eg: 1883 (if port 8883 is specified, a TLS connection will be established. see MQTT_GVL.TLS_DEFAULT_PORT)	*MQTT_CLIENT*
	sUsername	STRING	''	optional username, use empty string '' when not used	*MQTT_CLIENT*
	sPassword	STRING	''	optional password, use empty string '' when not used	*MQTT_CLIENT*
	sLastWillTopic	STRING	''	optional last will topic, use empty string '' when not used	*MQTT_CLIENT*
	sLastWillMessage	STRING	''	optional last will message body, use empty string '' when not used	*MQTT_CLIENT*
Output	xConnected	BOOL		HIGH: connection ready, LOW: not connected	*MQTT_CLIENT*
	error	*MQTT_ERROR*		see MQTT_ERROR	*MQTT_CLIENT*
	xDemoMode	BOOL	FALSE	HIGH: if running in 30 minutes demo mode (no license found)	*MQTT_CLIENT*
	xLicensed	BOOL	FALSE	HIGH: if license is valid, LOW: no valid license found	*MQTT_CLIENT*

MQTT_TLSCERT_CLIENT (FB)

FUNCTION_BLOCK MQTT_TLSCERT_CLIENT EXTENDS *MQTT_CLIENT*

Client connects to MQTT broker over secure connection (SSL/TLS) and uses locally installed certificate identified by passed certificate thumbprint to authenticate on server

Scope	Name	Type	Initial	Comment	Inherited from
Input	xConnect	BOOL		HIGH: connect to sBrokerIP and uiPort. LOW: idle, or disconnect existing connection	*MQTT_CLIENT*
	xAutoReConnect	BOOL	TRUE	HIGH: try every 1 second to reconnect after failure	*MQTT_CLIENT*
	sBrokerIP	STRING	'127.0.0.1'	Broker IP address (or hostname) as string, eg: '192.168.0.1'	*MQTT_CLIENT*
	uiPort	UINT	1883	Broker port, eg: 1883 (if port 8883 is specified, a TLS connection will be established. see MQTT_GVL.TLS_DEFAULT_PORT)	*MQTT_CLIENT*
	sUsername	STRING	''	optional username, use empty string '' when not used	*MQTT_CLIENT*
	sPassword	STRING	''	optional password, use empty string '' when not used	*MQTT_CLIENT*
	sLastWillTopic	STRING	''	optional last will topic, use empty string '' when not used	*MQTT_CLIENT*
	sLastWillMessage	STRING	''	optional last will message body, use empty string '' when not used	*MQTT_CLIENT*
Output	xConnected	BOOL		HIGH: connection ready, LOW: not connected	*MQTT_CLIENT*
	error	*MQTT_ERROR*		see MQTT_ERROR	*MQTT_CLIENT*
	xDemoMode	BOOL	FALSE	HIGH: if running in 30 minutes demo mode (no license found)	*MQTT_CLIENT*
	xLicensed	BOOL	FALSE	HIGH: if license is valid, LOW: no valid license found	*MQTT_CLIENT*
Input	sCertificateThumbprint	STRING		Thumb print of client certificate containing private key, for example: '5e572529a1633decc30639606e8db706650b4a9c'. Only certifiactes categorised as 'trusted' and 'own' are supported. Use CODESYS Security Agent (CODESYS store)	

Scope	Name	Type	Initial	Comment	Inherited from
				to create or manage certificates.	
Output	sCertificateIssuer	STRING		issuer of found client certificate	

MQTT_PUBLISH (FB)

FUNCTION_BLOCK MQTT_PUBLISH EXTENDS *MQTT_PUBLISH_IMPL*

Scope	Name	Type	Comment
Input	xPublish	BOOL	HIGH: rising edge starts publishing the message defined by sTopic, sPayload and qos
	pClient	POINTER TO *MQTT_CLIENT*	Pointer to MQTT_CLIENT
	sTopic	STRING(255)	Topic string, eg: '/test/temp1'
	qos	*MQTT_QOS*	see MQTT_QOS for supported QOS levels
Output	xPublished	BOOL	HIGH when QOS0 and message was sent successfully. HIGH when QOS1 and message was sent and acknowledged by broker successfully. Reset to LOW when xEnable is LOW
Input	sPayload	STRING(255)	Payload string, eg: '123.45'. Important: Payload cannot be longer than 255 characters. See MQTT_PUBLISH_EX for publishing longer strings or other data types.

MQTT_SUBSCRIBE (FB)

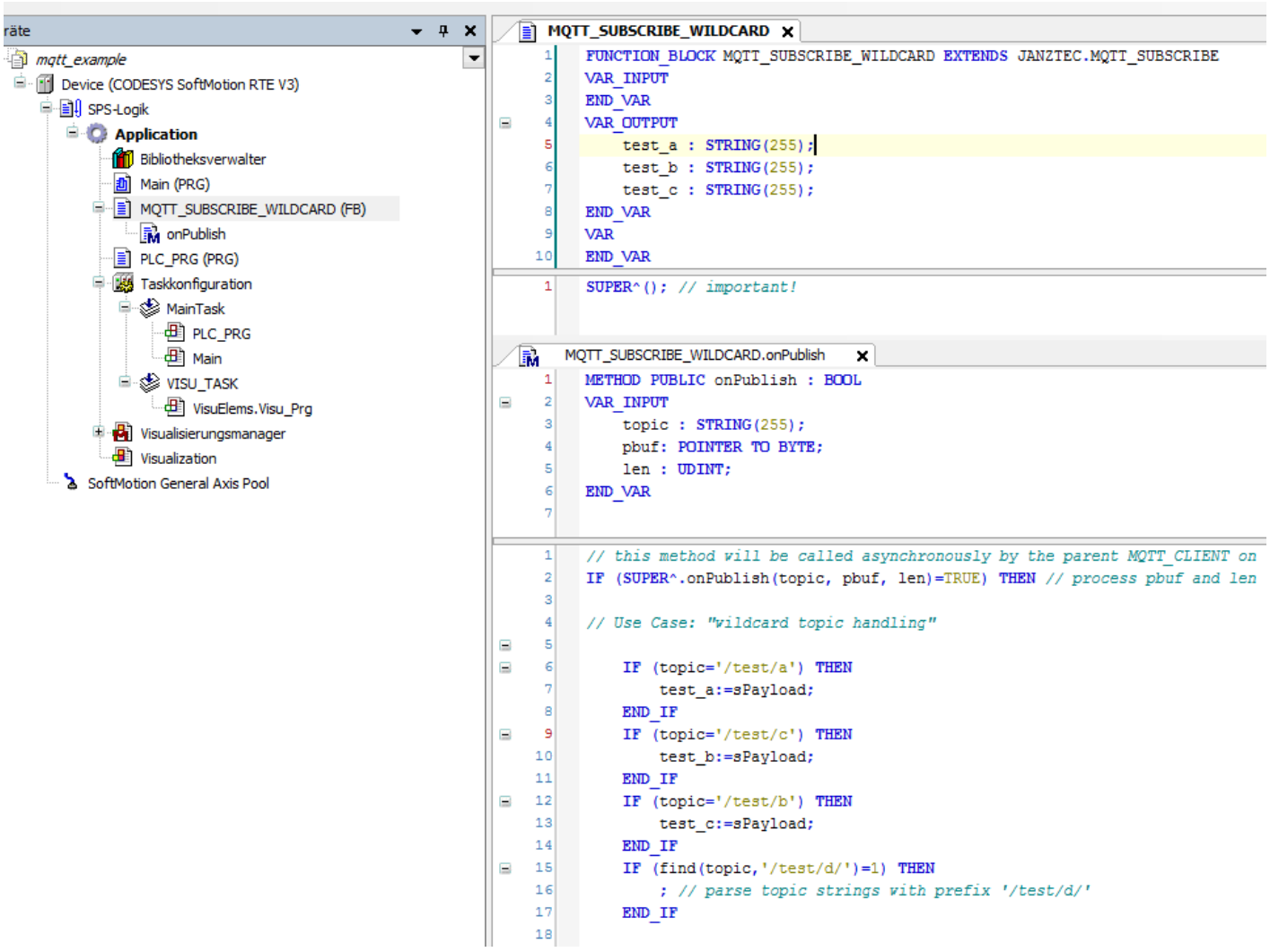
FUNCTION_BLOCK MQTT_SUBSCRIBE EXTENDS *MQTT_SUBSCRIBE_IMPL*

Scope	Name	Type	Initial	Comment
Input	xSubscribe	BOOL		HIGH starts subscribing to sTopic, Falling edge to LOW unsubscribes this sTopic
	xReceive	BOOL		Falling edge to LOW clears the output xReceived to LOW (does not actually prevent incoming messages to set sPayload)
	pClient	POINTER TO *MQTT_CLIENT*		Pointer to MQTT_CLIENT
	sTopic	STRING(255)		Topic string, eg: '/test/temp1' (change this variable only when xSubscribe is LOW)
Output	xSubscribed	BOOL		HIGH when subscription was successfully created and acknowledged by broker
	xReceived	BOOL		Rising to HIGH when new message was received. Falling to LOW when xReceive is LOW
	error	*MQTT_ERROR*	MQTT_ERROR.NO_ERROR	see MQTT_ERROR
	sPayload	STRING(255)		String containing the last reveived message payload (up to 254 characters and additional null-termination byte). See MQTT_SUBSCRIBE_EX for receiving longer strings or other data types.
	sLastTopic	STRING(255)		Topic string of last received message (useful if sTopic contains wildcards)

Use this function block to receive MQTT messages with string payload.

For this function block output variables sPayload and xReceived are handled once per call of the parent MQTT_CLIENT instance. So only the last matched message topic and payload is present in sLastTopic and sPayload. If you want to handle each incoming MQTT message manually, overwrite the onPublish method in a new function block inheriting from MQTT_SUBSCRIBE. (see screenshot below)

(see documentation in the example project included in this package)


räte
mqtt_example
Device (CODESYS SoftMotion RTE V3)
SPS-Logik
Application
Bibliotheksverwalter
Main (PRG)
MQTT_SUBSCRIBE_WILDCARD (FB)
onPublish
PLC_PRG (PRG)
Taskkonfiguration
MainTask
PLC_PRG
Main
VISU_TASK
VisuElems.Visu_Prg
Visualisierungsmanager
Visualization
SoftMotion General Axis Pool
MQTT_SUBSCRIBE_WILDCARD
FUNCTION_BLOCK MQTT_SUBSCRIBE_WILDCARD EXTENDS JANZTEC.MQTT_SUBSCRIBE
VAR_INPUT
END_VAR
VAR_OUTPUT
test_a : STRING(255);
test_b : STRING(255);
test_c : STRING(255);
END_VAR
VAR
END_VAR
SUPER^(); // important!
MQTT_SUBSCRIBE_WILDCARD.onPublish
METHOD PUBLIC onPublish : BOOL
VAR_INPUT
topic : STRING(255);
pbuf: POINTER TO BYTE;
len : UDINT;
END_VAR
// this method will be called asynchronously by the parent MQTT_CLIENT on
IF (SUPER^.onPublish(topic, pbuf, len)=TRUE) THEN // process pbuf and len
// Use Case: "wildcard topic handling"
IF (topic='/test/a') THEN
test_a:=sPayload;
END_IF
IF (topic='/test/c') THEN
test_b:=sPayload;
END_IF
IF (topic='/test/b') THEN
test_c:=sPayload;
END_IF
IF (find(topic,'/test/d/')=1) THEN
; // parse topic strings with prefix '/test/d/'
END_IF

MQTT_PUBLISH_EX (FB)

FUNCTION_BLOCK MQTT_PUBLISH_EX EXTENDS *MQTT_PUBLISH_IMPL*

Scope	Name	Type	Comment
Input	xPublish	BOOL	HIGH: rising edge starts publishing the message defined by sTopic, sPayload and qos
	pClient	POINTER TO *MQTT_CLIENT*	Pointer to MQTT_CLIENT
	sTopic	STRING(255)	Topic string, eg: '/test/temp1'
	qos	*MQTT_QOS*	see MQTT_QOS for supported QOS levels
Output	xPublished	BOOL	HIGH when QOS0 and message was sent successfully. HIGH when QOS1 and message was sent and acknowledged by broker successfully. Reset to LOW when xEnable is LOW
Input	pPayload	POINTER TO BYTE	Payload as pointer to byte
	udPayloadLength	UDINT	Payload length in bytes (see MQTT_GVL.MAX_PAYLOAD_SIZE for maximum allowed payload size)

MQTT_SUBSCRIBE_EX (FB)

FUNCTION_BLOCK MQTT_SUBSCRIBE_EX EXTENDS *MQTT_SUBSCRIBE_IMPL*

Scope	Name	Type	Initial	Comment
Input	xSubscribe	BOOL		HIGH starts subscribing to sTopic, Falling edge to LOW unsubscribes this sTopic
	xReceive	BOOL		Falling edge to LOW clears the output xReceived to LOW (does not actually prevent incoming messages to set sPayload)
	pClient	POINTER TO *MQTT_CLIENT*		Pointer to MQTT_CLIENT
	sTopic	STRING(255)		Topic string, eg: '/test/temp1' (change this variable only when xSubscribe is LOW)
Output	xSubscribed	BOOL		HIGH when subscription was successfully created and acknowledged by broker
	xReceived	BOOL		Rising to HIGH when new message was received. Falling to LOW when xReceive is LOW
	error	*MQTT_ERROR*	MQTT_ERROR.NO_ERROR	see MQTT_ERROR
	sLastTopic	STRING(255)		Topic string of last received message (useful if sTopic contains wildcards)
Input	pPayload	POINTER TO BYTE	0	Pointer where the payload will be saved (for output)
	udMaxPayloadLength	UDINT	0	Length of available memory in bytes for 'pPayload'. (see MQTT_GVL.MAX_PAYLOAD_SIZE for maximum allowed payload size) Example: When pPayload is pointer to DWORD variable, specify 4 (or sizeof()) for udMaxPayloadLength. (If udMaxPayloadLength is larger than received udPayloadLength, then the byte after the received payload is automatically cleared to 16#0 to allow (large) strings as payload. Can be disabled globally by setting MQTT_GVL.AUTO_TERMINATE_STRINGS_IN_SUBSCRIBE_EX:=false)
Output	udPayloadLength	UDINT	0	Length of payload saved to 'pPayload' in bytes. (see MQTT_GVL.MAX_PAYLOAD_SIZE for maximum allowed payload size)

Use this function block to receive MQTT messages with binary payload (pointer or array of bytes).

For this function block output is handled once per call of the parent MQTT_CLIENT instance.

So only the last matched message topic and payload is present in pPayload and sLastTopic. If you want to handle each incoming MQTT message manually, overwrite the onPublish method in a new function block inheriting from MQTT_SUBSCRIBE_EX.

(see documentation in the example project included in this package)

MQTT_GVL (GVL)

Name	Type	Initial	Comment
MAX_PAYLOAD_SIZE	UDINT	(1024 * 32)	Maximum message payload size. Default: 32KByte.
DEFAULT_TIMEOUT	UDINT	10000	default timeout in ms (default: 10sec)
AUTO_TERMINATE_STRINGS_IN_SUBSCRIBE_EX	BOOL	TRUE	see MQTT_SUBSCRIBE_EX for more information
CLIENT_RECONNECT_TIME	TIME	T#10S	Default time after client tries to reconnection to MQTT broker
CLIENT_CONNECT_TIMEOUT	TIME	T#10S	Default timeout after pending socket connections report connection error
PING_TIME	TIME	T#30S	Default time for cyclic sending of MQTT ping request to broker
VERIFY_TLS	BOOL	FALSE	**NO** verification of server certificate by default! To prevent this potential security risk (man in the middle attack), set this value manually to TRUE. Then make sure that the CA certificate is correctly installed and the system date and time is correct. Use CODESYS Security Agent (CODESYS store) to manage certificates.
TLS_DEFAULT_PORT	UINT	8883	Default port for witch automatically TLS connections will be established, even if MQTT_CLIENT is used instead of MQTT_TLS_CLIENT. Set to 0 if this features is not required.

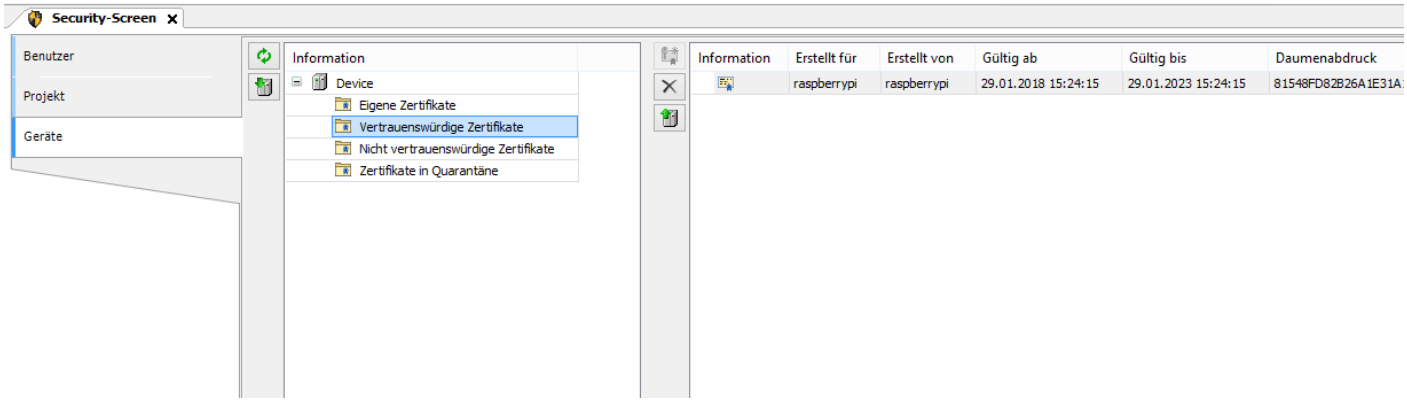
SSL/TLS Certificates

MQTT_TLS_CLIENT (FB)

Global variable MQTT_GVL.VERIFY_TLS is by default initialized with FALSE, so connections to the specified MQTT broker are encrypted but the broker certificate is not validated. This might be a potential security risk, because a Man-in-the-middle attack is possible! An attacker could then receive and decrypt all your sent MQTT messages, and send MQTT messages to you.

To prevent this attack, enable MQTT_GVL.VERIFY_TLS by setting the variable to TRUE before connecting to the your MQTT broker. This will validate the certificate of the broker and only if the certificate is valid the connection will be established. One requirement is, that the CA certificate of the broker is manually added to the “trusted certificate store” on your target device. You can use the CODESYS Security Agent (https://store.codesys.com/codesys-security-agent.html) to install this certificate to your device using your development environment.

Install certificate to *trusted certificate store*:



MQTT_TLSCERT_CLIENT (FB)

If you want to use a client certificate to authenticate with your MQTT broker, you need a client certificate on your target device which also contains the private key for this certificate.

At the time of this release, improvement request http://jira.codesys.com/browse/CDS-57009 is not included in a CODESYS release, so importing certificates containing private keys is not supported. Therefore, importing certificates with private keys created by major cloud IoT providers cannot be used.

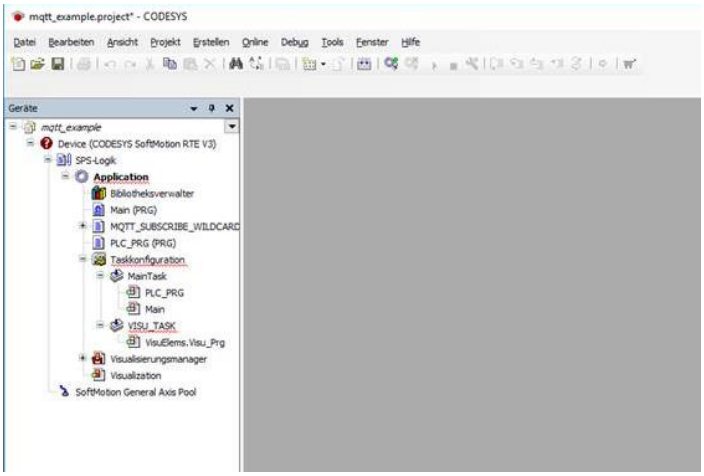
The second possibility would be to create a Certificate Signing Request on the target device (PLC-shell command “cert-createcsr”) and use this request to create a client certificate at the certificate authority. We were not able to use this method, because after importing the created client certificate, the private key which was used during creation of the CSR, is no longer available.

As a result, client authentication using certificates is not possible until CDS-57009 is available, or another possible exists which allows import of certificates containing the private key.

Troubleshooting

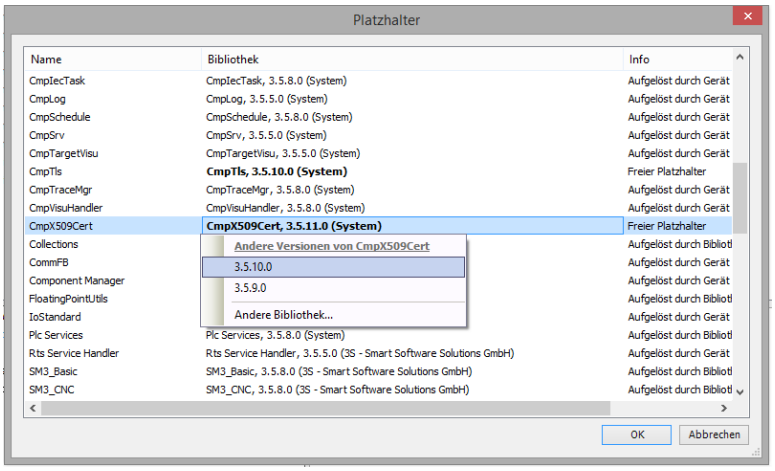
Example Project

The example project was created for CODESYS 3.5 SP8. If you use a newer version of the development environment, the device description for the 3.5 SP8 RTE is not installed by default and some libraries are not installed in the required versions.



Steps to resolve these problems:

1. Right click on the „Device (CODESYS SoftMotion RTE V3)“. Select „Update Device“ from the context menu and choose your target from the list, for example „CODESYS Control RTE V3“ (3.5.12.0)
2. Double click on the „Library Manager“. Click „Download missing libraries“ until no new libraries can be downloaded
3. If there are still libraries which are not correctly resolved, click „Placeholder“, search for entries with missing value in the colum „Library“. For example for CmpX509Cert click on the library colum and the select the latest version in the popup menu



TLS Library Problems

You can use the included library *MQTT_without_TLS.compiled-library* if you do not need TLS encryption and if there are unresolvable TLS/CmpX509 library dependency problems.

General Information

Supplier	Janz Tec AG Industrial Computing Architects Im Doerener Feld 8 33100 Paderborn - Germany
Support	Tel: +49 5251 1550-0 support@janztec.com
Product	Janz Tec MQTT library for CODESYS
Order Number	
Sales	CODESYS Store store.codesys.com
Scope of Delivery	▪ Library: „mqtt_library“ ▪ Sample Project: „mqtt_example“

System Requirements and Restrictions

Programming System	CODESYS Development System V3.5.8.10
Target System	CODESYS Control V3.5.8.10 – or newer
Supported Platforms / Devices	Notice: Use the sample project ‘mqtt_example.project’ to find out the supported features
Additional Requirements	Standard CODESYS libraries „TCP, SysSocket, CmpErrors“ must be supported on the target

Restrictions	• Broker must support MQTT protocol version 3.1.1
	• MQTT QoS level QoS2 is not supported
	• Topic names must be less than 120 characters
	• Encryption (SSL/TLS)
	  ○ Client authentication with certificate **not** supported! Wait for http://jira.codesys.com/browse/CDS-57009 see chapter SSL/TLS certificates
	  ○ Encryption is **not** supported on 64bit systems
	• WAGO e!COCKPIT is **not** supported!
	• WAGO Controller PFC100 is **not** supported!

Licensing	License activation optional on CODESYS Runtime Key or CODESYS Soft Key. License per runtime device necessary.
Required Accessory	CODESYS Security Key

Change History

Version	Description	Editor	Date
1.0	Initial Release for library version 1.0.0.1	ama	2016-09-07
1.1	Licensing changes	mde	2017-04-03
1.2	Added MQTT_CLIENT outputs	ama	2017-05-10
1.3	Added MQTT TLS & wildcard & last will support	ama	2018-01-03
1.4	Added MQTT_SUBSCRIBE custom message handling	ama	2018-01-05
2.0	MQTT_CLIENT and MQTT_TLS_CLIENT	ama	2018-01-29
2.0.0.4	Bug fixes: reconnection problem and publishing of empty messages	ama	2019-01-30
2.0.0.5	Bug fixes: MQTT ping/pong after reconnect	ama	2019-04-11
2.0.0.6	Bug fix in MQTT client connect timeout handling	ama	2019-04-30
2.0.0.7	Bug fix in MQTT reconnect	ama	2019-04-30
2.0.0.8	Replaced all String variable declarations with String(255)	ama	2019-09-24