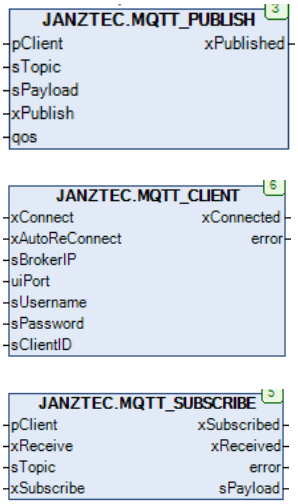
Datenblatt Janz Tec MQTT library for CODESYS

Produktbeschreibung

MQTT ist ein Client-Server-basiertes Publish/Subscribe-Nachrichtentransportprotokoll. Es ist schlank, offen, simpel und so entworfen, dass es leicht implementiert werden kann. Diese Eigenschaften machen es zum idealen Kommunikationsprotokoll in vielen Gebieten, beispielsweise in eingeschränkten Umgebungen wie Maschine-zu-Maschine-Kommunikation (M2M) und dem Internet der Dinge (IOT), in denen geringe Codegröße erforderlich ist und / oder die verfügbare Bandbreite eingeschränkt ist.

Diese von Janz Tec entwickelte CODESYS-Bibliothek implementiert einen Client für das MQTT-Protokoll. Mit ihrer Hilfe können Benutzer das MQTT-Protokoll auf einfache Art und Weise in ihren IEC-61131-Anwendungen nutzen. Die Bibliothek ist vollständig in IEC-61131Code geschrieben und damit unabhängig vom Zielsystem einsetzbar.

Funktionsumfang



Function Blocks

Features

- CODESYS MQTT-Client-Bibliothek (Protokollversion 3.1.1)
- Bibliothek ist vollständig in IEC-Code geschrieben, daher keine Abhängigkeit vom Zielsystem
- Publish: Beliebige Daten aus dem CODESYS-Projekt können als MQTT-Nachricht an einen Broker verschickt werden*
- Subscribe: Nachrichten von anderen MQTT-Brokern können empfangen und die Daten in CODESYS verwendet werden*
- Klassifikation von Nachrichten durch MQTT-Topics
- QoS-Levels
    - QoS0 (At-most-once)
    - QoS1 (At-least-once)
- Authentifizierung per Nutzername/Passwort
- Auto-Reconnect, falls die Verbindung unterbrochen wird
- Kanalverschlüsselung per SSL (TLS v1.1 Client)
- Last-Will-Funktionalität
- Topic-Wildcards (+/#)

* Die maximal zulässige Größe der Datenpakete ist in der globalen Variable MQTT_GVL.MAX_PAYLOAD_SIZE hinterlegt.

Zur Zeit nicht unterstützt

- QoS2 (Exactly once)

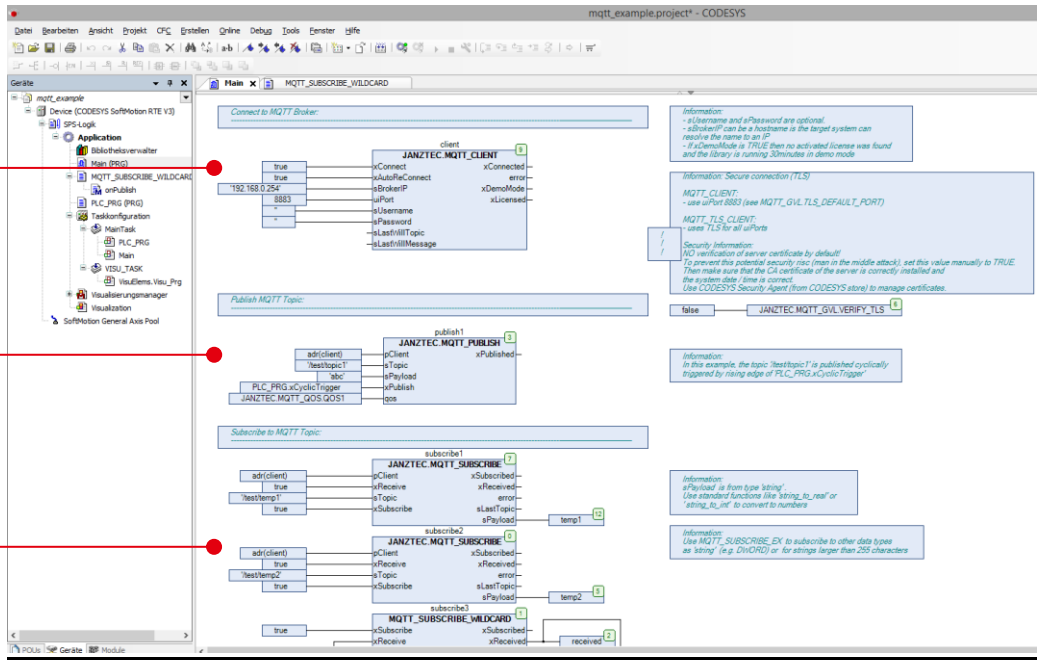
Produkt Screenshots

Beispielprojekt: „mqtt_example.project“

Mit MQTT Broker verbinden

Nachricht veröffentlichen

Für einen Topic registrieren



Nutzungsbeispiele

Schritt 1:

Konfigurieren Sie die IP-Adresse und Portnummer des MQTT Brokers im Beispielprojekt “mqtt_example” und starten Sie die Applikation.

Schritt 2:

Um die Beispielanzeige zu aktualisieren, verwenden Sie eine externe Anwendung um eine MQTT-Nachricht mit dem Topic „/test/temp4“ und dem Payload „12.00“ an Ihren Broker zu versenden.

Hinweis:

Falls Sie zurzeit noch keinen MQTT Broker verwenden, können Sie leicht einen Linux-basierten Rechner als MQTT Broker einsetzen, z.B. einen Raspberry Pi mit dem Raspbian Betriebssystem. Die folgenden Befehle installieren darauf den Mosquitto MQTT Broker.

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

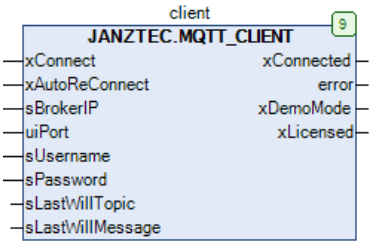
root@raspberrypi:/ service mosquitto start

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

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

Referenz

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)

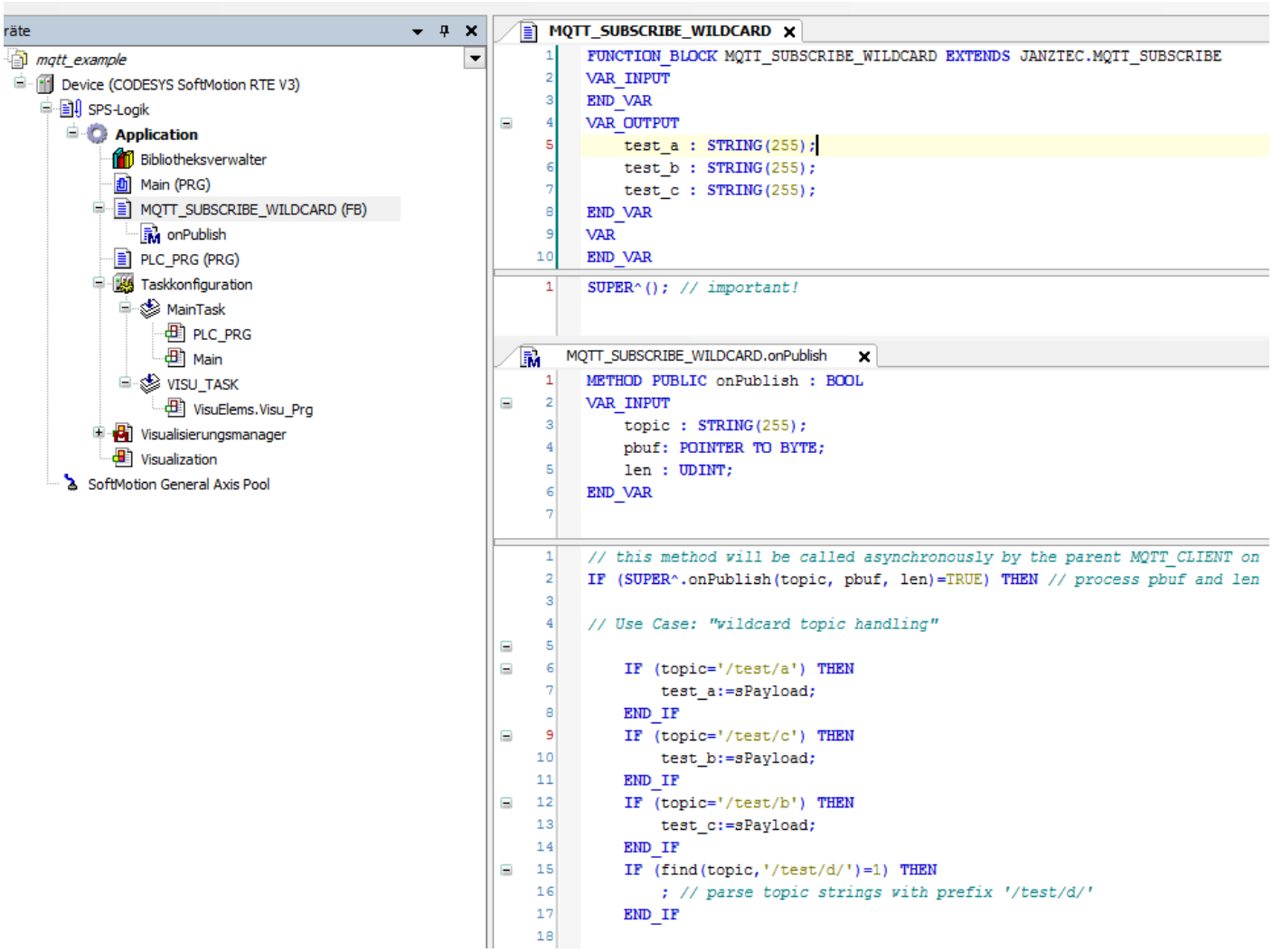
FUNCTION_BLOCK MQTT_SUBSCRIBE EXTENDS *MQTT_SUBSCRIBE_IMPL*

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)

Verwenden Sie diesen Funktionsbaustein, um MQTT-Nachrichten mit String-Nutzdaten zu empfangen.

Für diesen Funktionsblock werden die Ausgangsvariablen sPayload und xReceived einmal pro Aufruf der übergeordneten Instanz MQTTT_CLIENT verarbeitet. So ist in sLastTopic und sPayload nur das letzte übereinstimmende Message-Topic und Payload vorhanden. Wenn Sie jede eingehende MQTTT-Nachricht manuell behandeln möchten, überschreiben Sie die Methode onPublish in einem neuen Funktionsblock der von MQTT_SUBSCRIBE erbt. (Siehe Screenshot unten)

(siehe Dokumentation im Beispielprojekt, das in diesem Paket enthalten ist)


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)

Für diesen Funktionsblock wird die Ausgabe einmal pro Aufruf der übergeordneten Instanz MQTTT_CLIENT verarbeitet.

So ist in pPayload und sLastTopic nur das letzte übereinstimmende Nachrichten-Topic und die zugehörenden Nutzdaten verfügbar. Wenn Sie jede eingehende MQTT-Nachricht manuell behandeln möchten, überschreiben Sie die Methode onPublish in einem neuen Funktionsblock der von MQTT_SUBSCRIBE_EX erbt.

(siehe Dokumentation im Beispielprojekt, das in diesem Paket enthalten ist)

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.

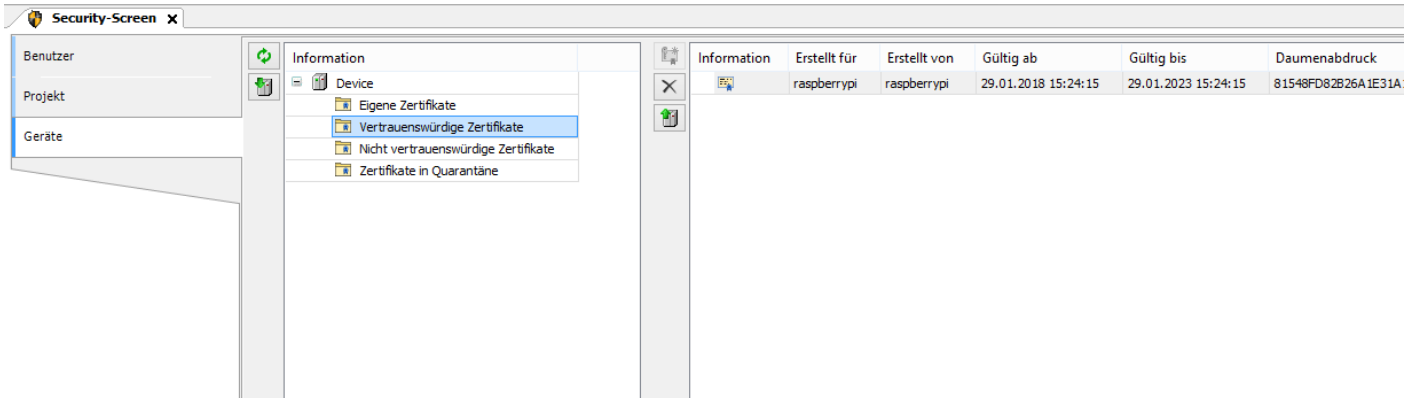
SSL/TLS Zertifikate

MQTT_TLS_CLIENT (FB)

Die globale Variable MQTT_GVL.VERIFY_TLS wird standardmäßig mit FALSE initialisiert, so dass Verbindungen zum angegebenen MQTT-Broker verschlüsselt werden, das Broker-Zertifikat jedoch nicht validiert wird. Dies könnte ein potentielles Sicherheitsrisiko darstellen, da ein Man-in-the-Middle-Angriff möglich ist! Ein Angreifer könnte alle Ihre gesendeten MQTT-Nachrichten empfangen und entschlüsseln und Ihnen MQTT-Nachrichten senden.

Um diesen Angriff zu verhindern, aktivieren Sie MQTT_GVL.VERIFY_TLS, indem Sie die Variable auf TRUE setzen, bevor Sie sich mit Ihrem MQTT Broker verbinden. Dies wird das Zertifikat des Brokers validieren und nur wenn das Zertifikat gültig ist, wird die Verbindung hergestellt. Voraussetzung ist, dass das CA-Zertifikat des Brokers manuell dem "Trusted Certificate Store" auf Ihrem Zielgerät hinzugefügt wird. Sie können den CODESYS Security Agent (https://store.codesys.com/codesys-security-agent.html) verwenden, um dieses Zertifikat mit Ihrer Entwicklungsumgebung auf Ihrem Gerät zu installieren.

Installieren von Zertifikaten in *vertrauenswürdige Zertifikate*



MQTT_TLSCERT_CLIENT (FB)

Wenn Sie sich mit einem Client-Zertifikat bei Ihrem MQTT-Broker authentifizieren wollen, benötigen Sie auf Ihrem Zielgerät ein Client-Zertifikat, das auch den privaten Schlüssel für dieses Zertifikat enthält.

Zum Zeitpunkt dieser Version ist der Verbesserungsvorschlag http://jira.codesys.com/browse/CDS-57009 nicht in einer CODESYS-Version enthalten, so dass aktuell der Import von Zertifikaten mit privaten Schlüsseln nicht unterstützt wird. Daher kann der Import von Zertifikaten mit privaten Schlüsseln, die von großen Cloud IoT-Providern erstellt wurden, nicht verwendet werden.

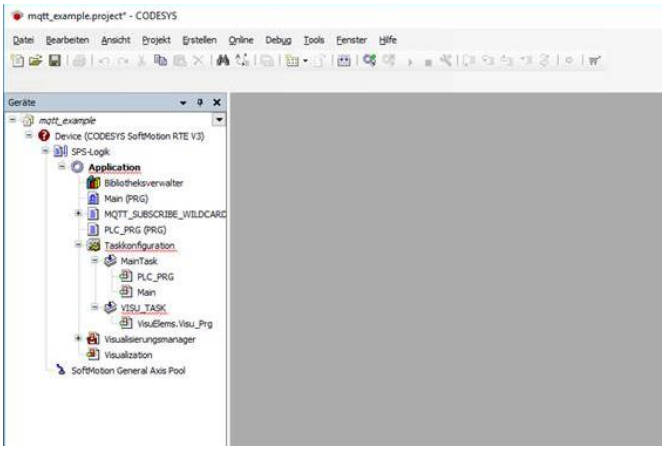
Die zweite Möglichkeit wäre, ein Certificate Signing Request auf dem Zielgerät zu erstellen (PLC-Shell-Befehl "cert-createcsr") und mit dieser Anforderung ein Client-Zertifikat bei der Zertifizierungsstelle zu erstellen. Wir konnten diese Methode bisher jedoch noch nicht erfolgreich anwenden, da nach dem Import des erstellten Client-Zertifikats der private Schlüssel, der bei der Erstellung des CSR verwendet wurde, nicht mehr verfügbar ist.

Daher ist eine Client-Authentifizierung mit Zertifikaten erst möglich, wenn CDS-57009 verfügbar ist, oder eine andere Möglichkeit besteht ein Clientcertifikat mit private Schlüssel auf einem Zielgerät zu installieren.

Problemlösung

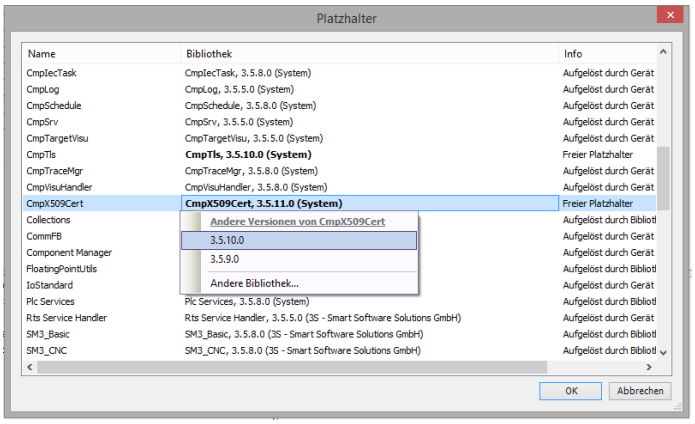
Beispielprojekt

Das Beispielprojekt wurde für CODESYS 3.5 SP8 erstellt. Wenn Sie eine neuere Version der Entwicklungsumgebung verwenden, ist die Gerätebeschreibung für die 3.5 SP8 RTE nicht standardmäßig installiert und auch einige Bibliotheken sind nicht in den benötigten Versionen installiert.



Schritte um die Probleme zu lösen

1. Klicken Sie mit der rechten Maustaste auf das "Gerät (CODESYS SoftMotion RTE V3)". Wählen Sie aus dem Kontextmenü "Gerät aktualisieren" und wählen Sie Ihr Zielsystemtyp aus der Liste aus, z.B. "CODESYS Control RTE V3" (3.5.12.0).
2. Doppelklicken Sie auf den "Bibliotheksverwalter". Klicken Sie auf "Download fehlender Bibliotheken", bis keine neuen Bibliotheken mehr heruntergeladen werden können.
3. Wenn es noch Bibliotheken gibt, die nicht korrekt aufgelöst sind, klicken Sie auf "Platzhalter", suchen Sie nach Einträgen mit fehlendem Wert in der Spalte "Bibliothek". Zum Beispiel für CmpX509Cert klicken Sie auf die Bibliotheksspalte und wählen Sie im Popup-Menü die neueste Version aus.



TLS Bibliotheksprobleme

Sie können die enthaltene Bibliothek *MQTT_without_TLS.compiled-library* verwenden, wenn Sie unauflösbare Bibliotheksreferenzen haben. Diese enthält keine Referenzen zur CmpTls und CmpdX509Cert Bibliothek.

Allgemeine Informationen

Hersteller	Janz Tec AG Industrial Computing Architects Im Dörener Feld 8 33100 Paderborn
Support	Tel: +49 5251 1550-0 support@janztec.com
Artikel	Janz Tec MQTT library for CODESYS
Artikelnummer	
Vertrieb	CODESYS Store store.codesys.com
Lieferumfang	▪ Library: „mqtt_library“ ▪ Sample Project: „mqtt_example“

Systemvoraussetzungen und Einschränkungen

Programmiersystem	CODESYS Development System V3.5.8.10
Laufzeitsystem	CODESYS Control V3.5.8.10 oder neuer
Unterstützte Plattformen / Geräte	Hinweis: Verwenden Sie das Beispiel ‘mqtt_example.project’, um die unterstützten Funktionen zu ermitteln.
Zusätzliche Anforderungen	Standard CODESYS Bibliotheken „TCP, SysSocket, CmpErrors“ müssen vom Zielsystem unterstützt werden.

Einschränkungen	• Broker muss MQTT Protokoll Version 3.1.1 unterstützen
	• MQTT QoS level 2 nicht unterstützt
	• Topic-Namen können maximal 120 Zeichen lang sein
	• Verschlüsselung (SSL/TLS)
	  ○ Clientauthentifizierung über Zertifikat wird derzeit **nicht** unterstützt! Warten auf http://jira.codesys.com/browse/CDS-57009 , siehe Kapitel SSL/TLS Zertifikate
	  ○ Auf 64Bit Zielsystemen wird Verschlüsselung **nicht** unterstützt
	• WAGO e!COCKPIT wird **nicht** unterstützt
	• WAGO Controller PFC100 wird **nicht** unterstützt

Lizenzierung	Lizenzaktivierung optional auf CODESYS Runtime Key oder CODESYS Soft Key. Lizenzierung pro Runtime (SPS) notwendig!
Erforderliches Zubehör	Ggf. 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	License 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
Product_MQTT_for_ CODESYS_datasheet 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