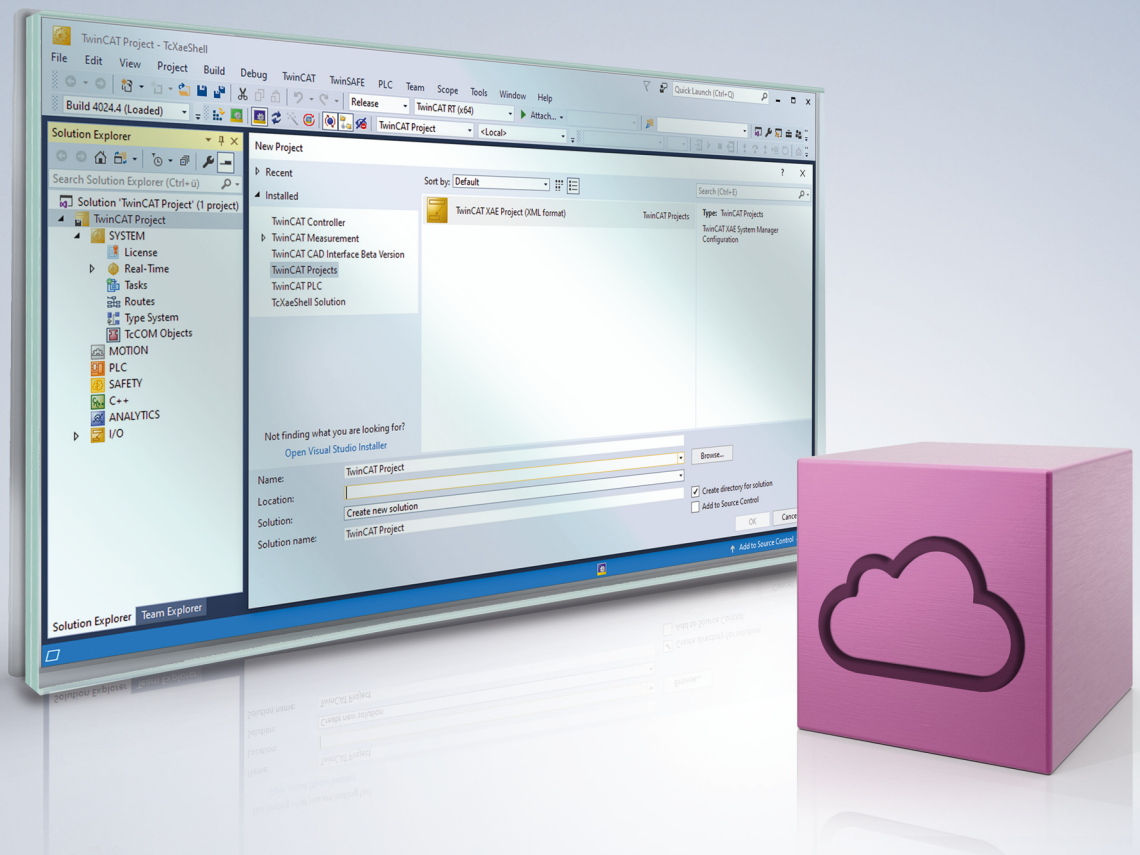
BECKHOFF New Automation Technology

Handbuch | DE

TF6701

TwinCAT 3 | IoT Communication (MQTT)



16.04.2024 | Version: 1.11.0

Inhaltsverzeichnis

1 Vorwort

1.1 Hinweise zur Dokumentation

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs- und Automatisierungstechnik, das mit den geltenden nationalen Normen vertraut ist.
Zur Installation und Inbetriebnahme der Komponenten ist die Beachtung der Dokumentation und der nachfolgenden Hinweise und Erklärungen unbedingt notwendig.
Das Fachpersonal ist verpflichtet, stets die aktuell gültige Dokumentation zu verwenden.

Das Fachpersonal hat sicherzustellen, dass die Anwendung bzw. der Einsatz der beschriebenen Produkte alle Sicherheitsanforderungen, einschließlich sämtlicher anwendbaren Gesetze, Vorschriften, Bestimmungen und Normen erfüllt.

Disclaimer

Diese Dokumentation wurde sorgfältig erstellt. Die beschriebenen Produkte werden jedoch ständig weiterentwickelt.
Wir behalten uns das Recht vor, die Dokumentation jederzeit und ohne Ankündigung zu überarbeiten und zu ändern.
Aus den Angaben, Abbildungen und Beschreibungen in dieser Dokumentation können keine Ansprüche auf Änderung bereits gelieferter Produkte geltend gemacht werden.

Marken

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® und XPlanar® sind eingetragene und lizenzierte Marken der Beckhoff Automation GmbH.
Die Verwendung anderer in dieser Dokumentation enthaltenen Marken oder Kennzeichen durch Dritte kann zu einer Verletzung von Rechten der Inhaber der entsprechenden Bezeichnungen führen.

Patente

Die EtherCAT-Technologie ist patentrechtlich geschützt, insbesondere durch folgende Anmeldungen und Patente:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
mit den entsprechenden Anmeldungen und Eintragungen in verschiedenen anderen Ländern.

EtherCAT® ist eine eingetragene Marke und patentierte Technologie lizenziert durch die Beckhoff Automation GmbH, Deutschland

Copyright

1.2 Zu Ihrer Sicherheit

Sicherheitsbestimmungen

Lesen Sie die folgenden Erklärungen zu Ihrer Sicherheit.
Beachten und befolgen Sie stets produktspezifische Sicherheitshinweise, die Sie gegebenenfalls an den entsprechenden Stellen in diesem Dokument vorfinden.

Haftungsausschluss

Die gesamten Komponenten werden je nach Anwendungsbestimmungen in bestimmten Hard- und Software-Konfigurationen ausgeliefert. Änderungen der Hard- oder Software-Konfiguration, die über die dokumentierten Möglichkeiten hinausgehen, sind unzulässig und bewirken den Haftungsausschluss der Beckhoff Automation GmbH & Co. KG.

Qualifikation des Personals

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs-, Automatisierungs- und Antriebstechnik, das mit den geltenden Normen vertraut ist.

Signalwörter

Im Folgenden werden die Signalwörter eingeordnet, die in der Dokumentation verwendet werden. Um Personen- und Sachschäden zu vermeiden, lesen und befolgen Sie die Sicherheits- und Warnhinweise.

Warnungen vor Personenschäden

⚠ GEFAHR

Es besteht eine Gefährdung mit hohem Risikograd, die den Tod oder eine schwere Verletzung zur Folge hat.

⚠ WARNUNG

Es besteht eine Gefährdung mit mittlerem Risikograd, die den Tod oder eine schwere Verletzung zur Folge haben kann.

⚠ VORSICHT

Es besteht eine Gefährdung mit geringem Risikograd, die eine mittelschwere oder leichte Verletzung zur Folge haben kann.

Warnung vor Umwelt- oder Sachschäden

HINWEIS

Es besteht eine mögliche Schädigung für Umwelt, Geräte oder Daten.

Information zum Umgang mit dem Produkt

Diese Information beinhaltet z. B.:
Handlungsempfehlungen, Hilfestellungen oder weiterführende Informationen zum Produkt.

1.3 Hinweise zur Informationssicherheit

Die Produkte der Beckhoff Automation GmbH & Co. KG (Beckhoff) sind, sofern sie online zu erreichen sind, mit Security-Funktionen ausgestattet, die den sicheren Betrieb von Anlagen, Systemen, Maschinen und Netzwerken unterstützen. Trotz der Security-Funktionen sind die Erstellung, Implementierung und ständige Aktualisierung eines ganzheitlichen Security-Konzepts für den Betrieb notwendig, um die jeweilige Anlage, das System, die Maschine und die Netzwerke gegen Cyber-Bedrohungen zu schützen. Die von Beckhoff verkauften Produkte bilden dabei nur einen Teil des gesamtheitlichen Security-Konzepts. Der Kunde ist dafür verantwortlich, dass unbefugte Zugriffe durch Dritte auf seine Anlagen, Systeme, Maschinen und Netzwerke verhindert werden. Letztere sollten nur mit dem Unternehmensnetzwerk oder dem Internet verbunden werden, wenn entsprechende Schutzmaßnahmen eingerichtet wurden.

Zusätzlich sollten die Empfehlungen von Beckhoff zu entsprechenden Schutzmaßnahmen beachtet werden. Weiterführende Informationen über Informationssicherheit und Industrial Security finden Sie in unserem https://www.beckhoff.de/secguide.

Die Produkte und Lösungen von Beckhoff werden ständig weiterentwickelt. Dies betrifft auch die Security-Funktionen. Aufgrund der stetigen Weiterentwicklung empfiehlt Beckhoff ausdrücklich, die Produkte ständig auf dem aktuellen Stand zu halten und nach Bereitstellung von Updates diese auf die Produkte aufzuspielen. Die Verwendung veralteter oder nicht mehr unterstützter Produktversionen kann das Risiko von Cyber-Bedrohungen erhöhen.

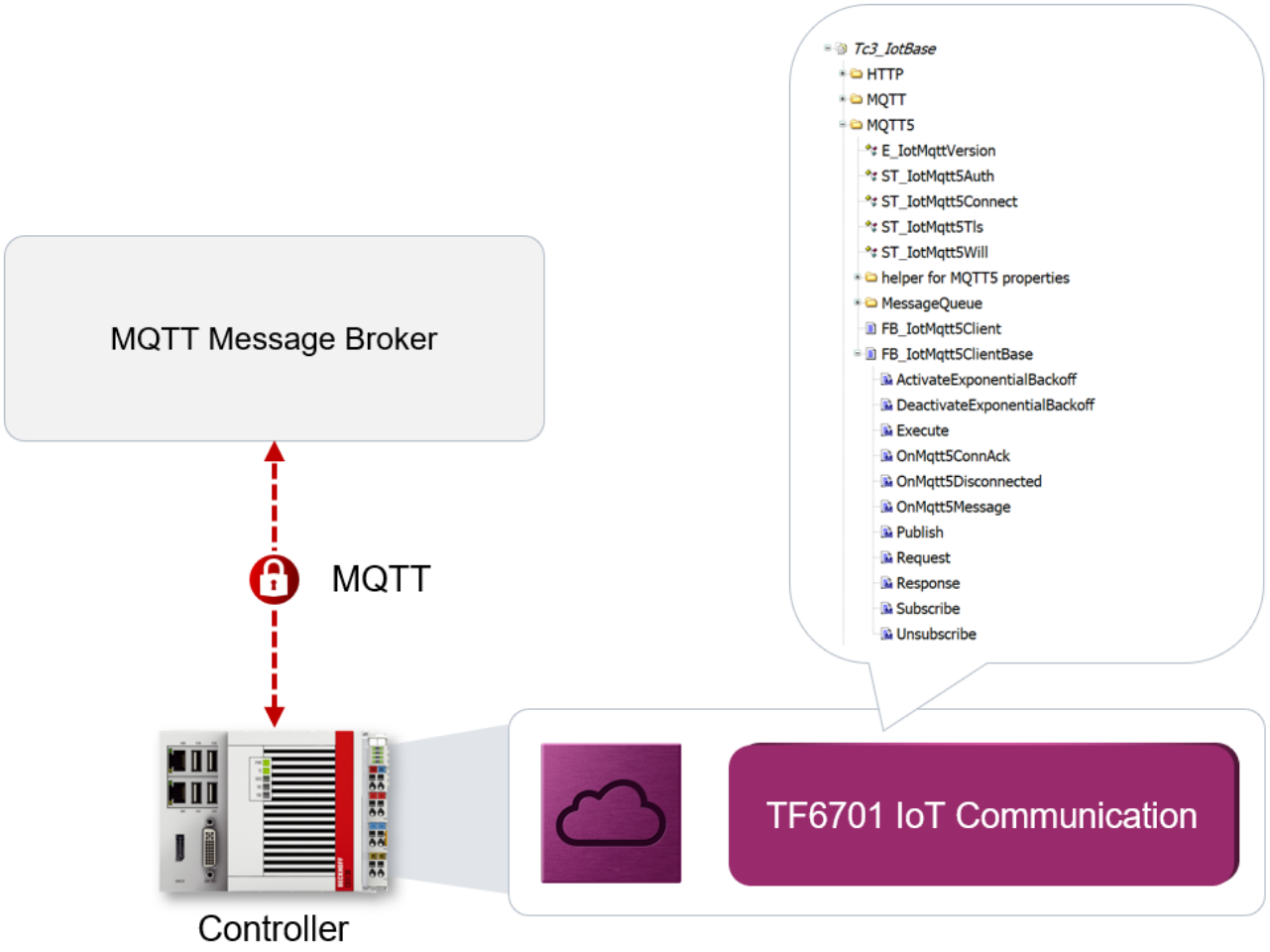
Um stets über Hinweise zur Informationssicherheit zu Produkten von Beckhoff informiert zu sein, abonnieren Sie den RSS Feed unter https://www.beckhoff.de/secinfo.

1.4 Ausgabestände der Dokumentation

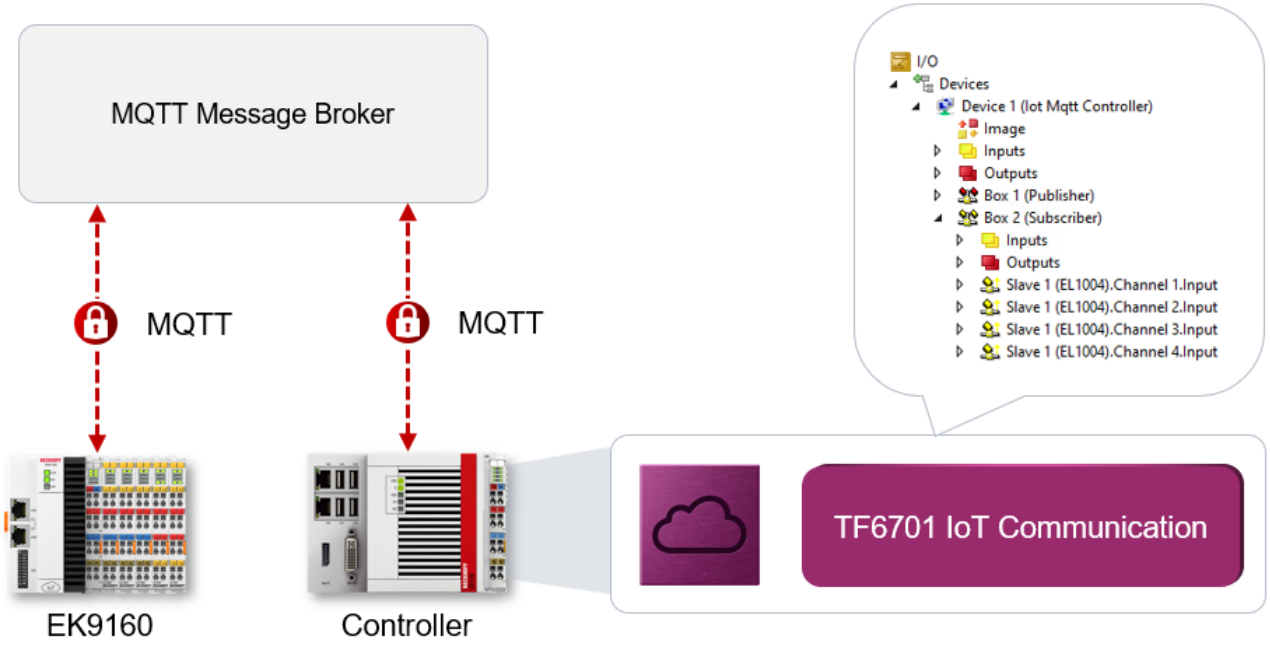
Version	Änderung
1.11.x	Installation/Installation [▶ 10]: Beschreibung der Installation unter TwinCAT 3.1 Build 4026 **Neu:** Getting Started [▶ 13] Unterstützte Funktionen [▶ 15] Unterstützte Message Broker [▶ 17] Protokollgrundlagen [▶ 27] Sicherheit [▶ 42] GetTimeSinceLastBrokerMessage [▶ 64] MQTT5 [▶ 71]

2 Übersicht

Mit den Funktionsbausteinen der SPS-Bibliothek Tc3_IotBase kann ein Publisher/Subscriber-basierter Datenaustausch zwischen der TwinCAT SPS und einem Message Broker über das Kommunikationsprotokoll MQTT erfolgen. Symbole können hierbei sowohl gesendet (Publish) als auch empfangen (Subscribe) werden. Das zu verwendende Datenformat ist hierbei frei definierbar und kann über zusätzliche SPS-Bibliotheken, z. B. die Tc3_JsonXml Bibliothek, erstellt werden.



Zusätzlich zu der SPS-Bibliothek steht ein TwinCAT I/O Gerät zur Verfügung, über welches sich eine Device-to-Device Kommunikationsverbindung über einen Message Broker konfigurieren lässt. Der Hauptanwendungsfall für dieses I/O Gerät ist, dass zwei TwinCAT-basierte Systeme Daten über einen Message Broker miteinander austauschen sollen. Alternativ kann hierüber auch ein EK9160 mit einem TwinCAT System über MQTT verbunden werden.



Produktkomponenten

TF6701 IoT Communication besteht aus den folgenden Komponenten, welche automatisch mit TwinCAT 3 installiert [▶ 10] werden:

- **Treiber**: TcIotDrivers.sys (ausgeliefert mit TwinCAT 3 XAE und XAR Setups)
- **SPS-Bibliothek**: Tc3_IotBase (ausgeliefert mit TwinCAT 3 XAE Setup)

3 Installation

3.1 Systemvoraussetzungen

Technische Daten	Beschreibung
Betriebssystem	Windows 7/10, Windows Embedded Standard 7, Windows CE 7, TwinCAT/BSD
Zielplattform	PC-Architektur (x86, x64 und ARM)
Minimale TwinCAT-Version (MQTTv3)	TwinCAT 3.1 Build 4022.0 und höher
Minimale TwinCAT-Version (MQTTv5)	TwinCAT 3.1 Build 4026.0 und höher
Erforderliches TwinCAT-Setup-Level	TwinCAT 3 XAE, XAR
Erforderliche TwinCAT-Lizenz	TF6701 TC3 IoT Communication

Bitte beachten Sie auch unseren Dokumentationsartikel zu den unterstützten MQTT-Funktionen [▶ 15] für eine feingranulare Übersicht.

3.2 Installation

TwinCAT 3.1 Build 4022 und 4024

Alle benötigten Komponenten werden direkt mit dem TwinCAT Setup ausgeliefert.

- TwinCAT XAE Setup: Enthält den MQTT-Treiber und die SPS-Bibliothek (Tc3_IotBase [▶ 57])
- TwinCAT XAR Setup: Enthält nur den MQTT-Treiber

TwinCAT 3.1 Build 4026

Der MQTT-Treiber ist bereits im TwinCAT Standard Workload enthalten. Die SPS-Bibliothek Tc3_IotBase kann über das Package TwinCAT.XAE.PLC.Lib.Tc3_IotBase installiert werden.

Wenn Sie TwinCAT 3.1 Build 4026 (und höher) auf dem Betriebssystem Microsoft Windows verwenden, können Sie diese Function über den TwinCAT Package Manager installieren, siehe Dokumentation zur Installation.

Normalerweise installieren Sie die Function über den entsprechenden Workload; dennoch können Sie das im Workload enthaltene Paket auch einzeln installieren. Diese Dokumentation beschreibt im Folgenden kurz den Installationsvorgang über den Workload.

Kommandozeilenprogramm TcPkg

Über das TcPkg **C**ommand **L**ine **I**nterface (CLI) können Sie sich die verfügbaren Workloads auf dem System anzeigen lassen:

```
tcpkg list -t workload
```

Über das folgende Kommando können Sie den Workload der TF6701 IoT Communication MQTT-Function installieren.

```
tcpkg install TF6701.IotCommunication.XAE
```

TwinCAT Package Manager UI

Über das **U**ser **I**nterface (UI) können Sie sich alle verfügbaren Workloads anzeigen lassen und diese bei Bedarf installieren.
Folgen Sie hierzu den entsprechenden Anweisungen in der Oberfläche.

3.3 Lizenzierung

Die TwinCAT 3 Function ist als Vollversion oder als 7-Tage-Testversion freischaltbar. Beide Lizenztypen sind über die TwinCAT-3-Entwicklungsumgebung (XAE) aktivierbar.

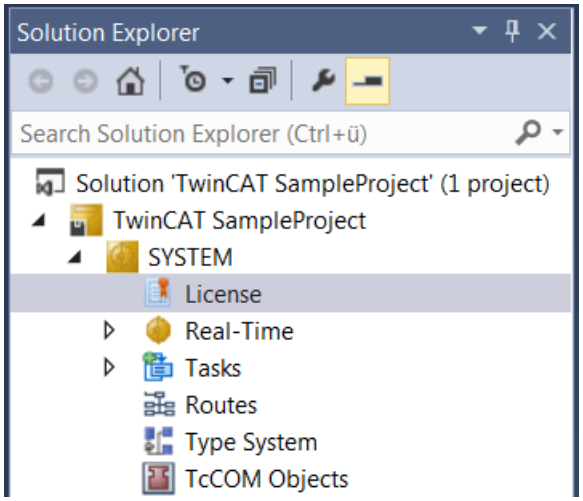
Lizenzierung der Vollversion einer TwinCAT 3 Function

Die Beschreibung der Lizenzierung einer Vollversion finden Sie im Beckhoff Information System in der Dokumentation „TwinCAT-3-Lizenzierung“.

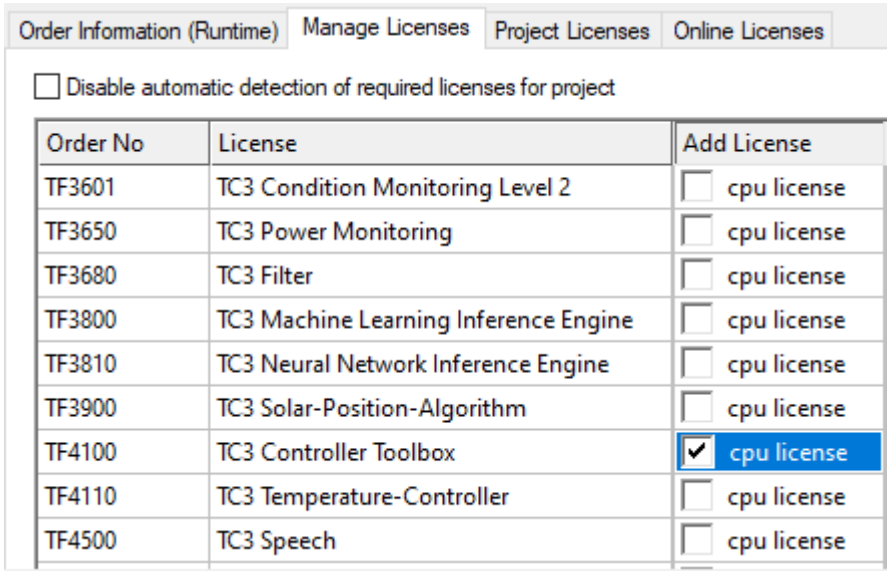
Lizenzierung der 7-Tage-Testversion einer TwinCAT 3 Function

Eine 7-Tage-Testversion kann nicht für einen TwinCAT-3-Lizenz-Dongle freigeschaltet werden.

1. Starten Sie die TwinCAT-3-Entwicklungsumgebung (XAE).
2. Öffnen Sie ein bestehendes TwinCAT-3-Projekt oder legen Sie ein neues Projekt an.
3. Wenn Sie die Lizenz für ein Remote-Gerät aktivieren wollen, stellen Sie das gewünschte Zielsystem ein. Wählen Sie dazu in der Symbolleiste in der Drop-down-Liste **Choose Target System** das Zielsystem aus.
   ⇨ Die Lizenzierungseinstellungen beziehen sich immer auf das eingestellte Zielsystem. Mit der Aktivierung des Projekts auf dem Zielsystem werden automatisch auch die zugehörigen TwinCAT-3-Lizenzen auf dieses System kopiert.
4. Klicken Sie im **Solution Explorer** im Teilbaum **SYSTEM** doppelt auf **License.**

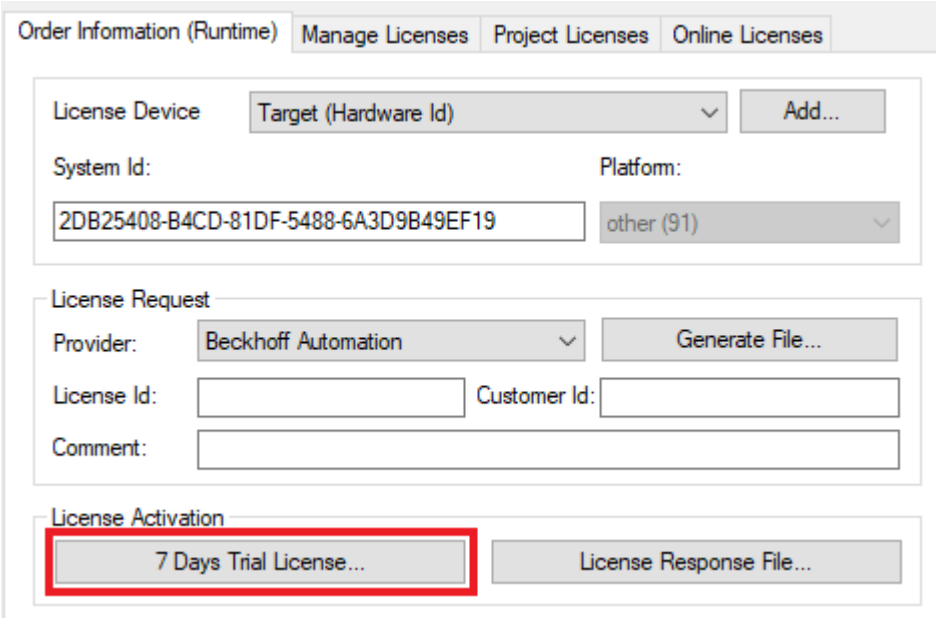


   ⇨ Der TwinCAT-3-Lizenzmanager öffnet sich.
5. Öffnen Sie die Registerkarte **Manage Licenses.** Aktivieren Sie in der Spalte **Add License** das Auswahlkästchen für die Lizenz, die Sie Ihrem Projekt hinzufügen möchten (z. B. „TF4100 TC3 Controller Toolbox“).

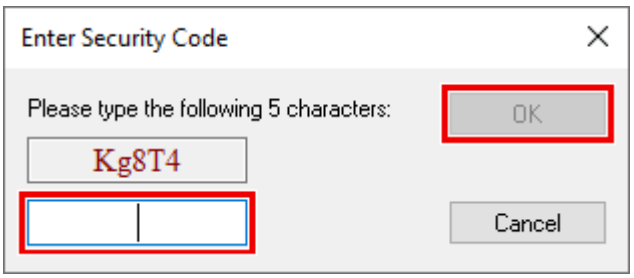


Order No	License	Add License
TF3601	TC3 Condition Monitoring Level 2	cpu license
TF3650	TC3 Power Monitoring	cpu license
TF3680	TC3 Filter	cpu license
TF3800	TC3 Machine Learning Inference Engine	cpu license
TF3810	TC3 Neural Network Inference Engine	cpu license
TF3900	TC3 Solar-Position-Algorithm	cpu license
TF4100	TC3 Controller Toolbox	cpu license
TF4110	TC3 Temperature-Controller	cpu license
TF4500	TC3 Speech	cpu license

6. Öffnen Sie die Registerkarte **Order Information (Runtime).**
   ⇨ In der tabellarischen Übersicht der Lizenzen wird die zuvor ausgewählte Lizenz mit dem Status „missing“ angezeigt**.**
7. Klicken Sie auf **7 Days Trial License...**, um die 7-Tage-Testlizenz zu aktivieren.



   ⇨ Es öffnet sich ein Dialog, der Sie auffordert, den im Dialog angezeigten Sicherheitscode einzugeben.



8. Geben Sie den Code genauso ein, wie er angezeigt wird, und bestätigen Sie ihn.
9. Bestätigen Sie den nachfolgenden Dialog, der Sie auf die erfolgreiche Aktivierung hinweist.
   ⇨ In der tabellarischen Übersicht der Lizenzen gibt der Lizenzstatus nun das Ablaufdatum der Lizenz an.
10. Starten Sie das TwinCAT-System neu.

⇨ Die 7-Tage-Testversion ist freigeschaltet.

4 Technische Einführung

4.1 Getting Started

Dieser Dokumentationsartikel soll Ihnen einen ersten, schnellen Start in die Verwendung dieses Produkts ermöglichen. Nach der erfolgreichen Installation [▶ 10] und Lizenzierung [▶ 10] führen Sie die folgenden Schritte aus, um eine Verbindung zu einem MQTT Message Broker herzustellen und Nachrichten zu senden und zu empfangen.

> **Message Broker Installation**
>
> Dieser Dokumentsartikel setzt das Vorhandensein eines lokal installierten und funktionsfähigen MQTT Message Brokers voraus. Als Beispiel verwenden wir hier den Mosquitto Message Broker, sie können aber jeden beliebigen Message Broker verwenden.

Einrichtung des Message Brokers

Der Mosquitto Message Broker wird seit der Version 2.x mit einer Konfiguration ausgeliefert, die die Einrichtung von Security Maßnahmen erfordert, damit ein sicherer Betrieb des Message Brokers gewährleistet wird. Im Folgenden zeigen wir Ihnen, wie Sie die Konfiguration des Mosquitto Message Brokers so abändern, dass eine ungesicherte Kommunikationsverbindung mit dem Broker hergestellt werden kann. Dies soll jedoch ausschliesslich zu Testzwecken in einer vertrauenswürdigen Betriebsumgebung erfolgen. Für den produktiven Betrieb empfehlen wir die Verwendung einer sicheren Broker-Konfiguration.

1. Installieren Sie den Mosquitto Message Broker auf Ihrem System.
2. Erstellen Sie ein Backup der Datei mosquitto.conf aus dem Mosquitto Installationsverzeichnis. Dieses befindet sich typischerweise unter C:\Program Files\mosquitto.
3. Öffnen Sie die Datei mosquitto.conf mit einem Texteditor Ihrer Wahl und entfernen Sie den vorhandenen Inhalt. Fügen Sie den folgenden Inhalt hinzu und speichern Sie die Datei.

```
listener 1883
allow_anonymous true
```

4. Starten Sie den Mosquitto Message Broker neu, entweder über den entsprechenden Windows-Dienst oder manuell über die Konsole bzw. die mosquitto.exe.

⇨ Sie haben nun den Mosquitto Message Broker so konfiguriert, dass dieser auf eingehende Clientverbindung auf Port 1883 lauscht und keinerlei Security (weder eine Benutzerauthentifizierung, noch Client-Zertifikate) benötigt.

Sie können nun mit der Einrichtung des TwinCAT Projekts beginnen.

Einrichtung des TwinCAT Projekts

Nachdem Sie den Message Broker installiert haben, können Sie nun mit der Einrichtung des TwinCAT Projekts beginnen, um eine Verbindung zum Broker aufzubauen und Nachrichten über diesen auszutauschen.

1. Erstellen Sie ein TwinCAT-Projekt mit einer SPS und fügen Sie die Tc3_IotBase als Bibliotheksreferenz hinzu.
2. Legen Sie einen Programmbaustein an und erstellen Sie im **Deklarationsteil** eine Instanz von FB_IotMqttClient [▶ 57] sowie zwei Hilfsvariablen, um den Programmablauf bei Bedarf steuern zu können.

```
PROGRAM PrgMqttCom
VAR
    fbMqttClient    : FB_IotMqttClient;
    bSetParameter   : BOOL := TRUE;
    bConnect        : BOOL := TRUE;
END_VAR
```

3. Deklarieren Sie für die zu verschickende MQTT-Nachricht zwei Variablen für Topic und Payload. Im Beispiel soll jede Sekunde eine Nachricht verschickt werden, was über einen entsprechenden Timer-Baustein realisiert werden soll. Eine Zählervariable soll später für den Nachrichteninhalt verwendet werden.

```
    (* published message *)
    sTopicPub   : STRING(255) := 'MyTopic';
    sPayloadPub : STRING(255);
    fbTimer : TON := (PT:=T#1S);
    i : UDINT;
```

4. Deklarieren Sie für den Nachrichtenempfang eine Variable, die das zu empfangende Topic enthält und zwei weitere Variablen, die Topic und Payload der zuletzt empfangenen Nachricht anzeigen. Die empfangenen Nachrichten sollen in einer Warteschlange (Queue) gesammelt werden, um sequenziell ausgewertet werden zu können. Hierfür deklarieren Sie eine Instanz von FB_IotMqttMessageQueue [▶ 66] sowie eine Instanz von FB_IotMqttMessage [▶ 68].

```
    (* received message *)
    bSubscribed    : BOOL;
    sTopicSub      : STRING(255) := 'MyTopic';
    {attribute 'TcEncoding':='UTF-8'}
    sTopicRcv      : STRING(255);
    {attribute 'TcEncoding':='UTF-8'}
    sPayloadRcv    : STRING(255);
    fbMessageQueue : FB_IotMqttMessageQueue;
    fbMessage      : FB_IotMqttMessage;
```

5. Im **Programmteil** muss der MQTT Client zyklisch getriggert werden, um den Verbindungsaufbau zum Broker, den Verbindungserhalt und den Nachrichtenempfang zu gewährleisten. In diesem Beispiel werden die Verbindungsparameter einmalig vor dem Client-Aufruf im Programmcode zugewiesen. Weil dies generell meist nur einmal notwendig ist, könnten die Parameter aber auch bereits im Deklarationsteil bei der Instanziierung des MQTT Client angegeben werden. Nicht alle Parameter müssen zugewiesen werden. In unserem Beispiel ist der Broker lokal auf demselben System installiert auf dem Sie auch das SPS Projekt aktivieren wollen. Alternativ können Sie auch die IP-Adresse oder den Hostnamen des Systems angeben, auf dem der Message Broker installiert wurde.

```
IF bSetParameter THEN
    bSetParameter               := FALSE;
    fbMqttClient.sHostName      := 'localhost';
    fbMqttClient.nHostPort      := 1883;
    fbMqttClient.ipMessageQueue := fbMessageQueue;
END_IF
fbMqttClient.Execute(bConnect);
```

6. Der zyklische Aufruf vom MQTT Client sorgt für den Empfang der Nachrichten. Der Client empfängt alle Nachrichten, auf deren Topic er sich zuvor beim Broker angemeldet hat (siehe nächster Schritt) und legt diese in der Message Queue ab. Sobald Nachrichten verfügbar sind, rufen Sie die Methode Dequeue() auf und erhalten über das Nachrichtenobjekt fbMessage Zugriff auf die Nachrichteneigenschaften wie Topic und Payload. Bei der folgenden Implementierung der Nachrichtenauswertung wird pro Zyklus eine empfangene Nachricht ausgewertet. Falls mehrere Nachrichten in der Message-Queue gesammelt wurden, verteilt sich deren Auswertung entsprechend auf mehrere Zyklen.

```
IF fbMessageQueue.nQueuedMessages > 0 THEN
    IF fbMessageQueue.Dequeue(fbMessage:=fbMessage) THEN
        fbMessage.GetTopic(pTopic:=ADR(sTopicRcv), nTopicSize:=SIZEOF(sTopicRcv) );
        fbMessage.GetPayload(pPayload:=ADR(sPayloadRcv), nPayloadSize:=SIZEOF(sPayloadRcv), bSet
NullTermination:=FALSE);
    END_IF
END_IF
```

7. Sobald die Verbindung zum Broker aufgebaut wird, soll sich der Client auf ein bestimmtes Topic anmelden. Ebenso soll jede Sekunde eine Nachricht versandt werden. In diesem Beispiel ist `sTopicPub = sTopicSub`, sodass ein Loopback entsteht und die gesendeten Nachrichten gleich wieder empfangen werden. In anderen Applikationen unterscheiden sich die Topics typischerweise.

```
IF fbMqttClient.bConnected THEN
    IF NOT bSubscribed THEN
        bSubscribed := fbMqttClient.Subscribe(sTopic:=sTopicSub, eQoS:=TcIotMqttQos.AtMostOnceDe
livery);
    END_IF
    fbTimer(IN:=TRUE);
    IF fbTimer.Q THEN // publish new payload every second
        fbTimer(IN:=FALSE);
        i := i + 1;
        sPayloadPub := CONCAT('MyMessage', TO_STRING(i));
        fbMqttClient.Publish(sTopic:= sTopicPub,
pPayload:= ADR(sPayloadPub), nPayloadSize:= LEN2(ADR(sPayloadPub))+1,
eQoS:= TcIotMqttQos.AtMostOnceDelivery, bRetain:= FALSE, bQueue:= FALSE );
    END_IF
END_IF
```

Nächste Schritte

Nachdem Sie nun erfolgreich eine Verbindung zum Message Broker hergestellt haben, empfehlen wir Ihnen den Mosquitto Message Broker wieder auf seinen Auslieferungszustand zurückzusetzen. Hierfür nehmen Sie bitte die Backupdatei der mosquitto.conf, welche Sie in den vorhergehenden Schritten erstellt haben. Diese Datei ist, zusammen mit der Dokumentation des Brokers, eine gute Grundlage für weitere Schritte, z.B. zur Einrichtung einer sicheren Message Broker Betriebsumgebung unter Berücksichtigung von Client/ Server Zertifikaten und einer Benutzerauthentifizierung.

In obigem Beispiel haben wir als Payload der Nachricht eine String-Variable verschickt bzw. empfangen. Da das Datenformat [▶ 30] bei MQTT nicht spezifiziert ist, können Sie frei entscheiden, wie dieses aufgebaut sein soll. Typische IoT-Anwendungen verwenden jedoch ein JSON- oder XML-basiertes Datenformat. Hierfür stellt Beckhoff die SPS-Bibliothek Tc3_JsonXml bereit, mit deren Hilfe Sie JSON- und XML-Dokumente erstellen und auslesen können.

4.2 Unterstützte Funktionen

Die folgende Tabelle gibt einen Überblick über alle unterstützten Funktionen. Generell werden sowohl MQTT 3.1.1 (im Folgenden als MQTT3 bezeichnet) als auch MQTT 5.0 (im Folgenden als MQTT5 bezeichnet) unterstützt.

Funktion	Unterstützt in	Min. TwinCAT Version	Beschreibung
Benutzerauthentifizierung	MQTT3/MQTT5	3.1 Build 4022.0	Eine Benutzername-/Passwort-Kombination kann zur Anmeldung am Message Broker verwendet werden.
Cipher Suites	MQTT3/MQTT5	3.1 Build 4022.0	Angabe der zu verwendenden Cipher Suites.
Clean Session	MQTT3/MQTT5	3.1 Build 4022.0	Die Verwendung von CleanSession wird aktuell nur für den Fall CleanSession = TRUE unterstützt.
Client-ID	MQTT3/MQTT5	3.1 Build 4022.0	Angabe einer MQTT Client-ID.
Connection acknowledgement	MQTT5	3.1 Build 4026.0	Das Connection Acknowledgement [▶ 36] Feature kann verwendet werden.
Datenformat	MQTT3/MQTT5	3.1 Build 4022.0	Das Datenformat [▶ 30] kann frei definiert werden. Entsprechende Hilfsbibliotheken zur einfachen Verwendung von JSON und XML stehen zur Verfügung.
Keep Alive	MQTT3/MQTT5	3.1 Build 4022.0	Setzen des KeepAlive Werts für die MQTT Verbindung.
Last Will	MQTT3/MQTT5	3.1 Build 4022.0	Definition einer Last Will [▶ 34] Nachricht.
Message expiry interval	MQTT5	3.1 Build 4026.0	Das Message expiry interval [▶ 40] kann gesetzt werden.
Pre-Shared Key (PSK)	MQTT3/MQTT5	3.1 Build 4022.0	Ein PSK kann zur Absicherung des Transportkanals (mittels TLS) verwendet werden.
Publish	MQTT3/MQTT5	3.1 Build 4022.0	Versenden (publishen) von Nachrichten an ein Topic.
QoS 0, 1, 2	MQTT3/MQTT5	3.1 Build 4022.0	Die QoS [▶ 31]-Level 0, 1 und 2 können bei Publish- und Subscribe-Vorgängen verwendet werden.
Reason codes	MQTT5	3.1 Build 4026.0	Reason codes [▶ 41] können empfangen und ausgewertet werden.
Request/Response	MQTT5	3.1 Build 4026.0	Senden/Empfangen von Nachrichten basierend auf dem Request/Response Verfahren [▶ 34].
Retain	MQTT3/MQTT5	3.1 Build 4022.0	Nachrichten können beim Publish-Vorgang mit dem Retain [▶ 33]-Flag versehen werden.

Funktion	Unterstützt in	Min. TwinCAT Version	Beschreibung
Session expiry interval	MQTT5	3.1 Build 4026.0	Das Session expiry interval [▶ 40] kann gesetzt und verwendet werden.
Subscribe/Unsubscribe	MQTT3/MQTT5	3.1 Build 4022.0	Erzeugen einer Subscription auf ein Topic zum Empfangen von Nachrichten.
Topics	MQTT3/MQTT5	3.1 Build 4022.0	Topic [▶ 28]-Hierarchien können frei definiert werden.
Transport Layer Security (TLS) Version 1.1, 1.2 und 1.3	MQTT3/MQTT5	3.1 Build 4022.0	TLS kann zur Absicherung des Transportkanals verwendet werden.
User Properties	MQTT5	3.1 Build 4026.0	Verwendung von User Properties [▶ 35] zur Definition von Metadaten an einer Nachricht.
Wildcard Subscriptions	MQTT3/MQTT5	3.1 Build 4022.0	Verwendung von Wildcards (#, +) für eine Subscription.
Zertifikate	MQTT3/MQTT5	3.1 Build 4022.0	Zertifikate können zur Absicherung des Transportkanals (mittels TLS) verwendet werden. Hierbei wird das PEM-Format verwendet.
Zertifikatssperrlisten	MQTT3/MQTT5	3.1 Build 4022.0	Verwendung von Zertifikatssperrlisten.

4.3 Unterstützte Message Broker

MQTT ist ein standardisiertes Transportprotokoll, welches in den vergangenen Jahren eine große Verbreitung gefunden hat. Gerade im Bereich der Cloud-Konnektivität setzen alle großen Cloudanbieter für die Anbindung von IoT-Geräten mittlerweile auf MQTT als Transportprotokoll. Die folgende Tabelle gibt einen Überblick über gängige Cloud-Dienste oder Softwareapplikationen mit einer MQTT-Schnittstelle, welche bereits in verschiedenen Kundenanwendungen Verwendung gefunden haben. Diese Tabelle stellt keine vollständige Auflistung aller unterstützten Message Broker dar, sondern nur die gängigsten Systeme welche oft in Kundenapplikationen Anwendung finden.

Cloud-Dienst oder Software	Beschreibung
AWS IoT Core	AWS IoT Core ist ein von Amazon Web Services als nativer Cloud-Service verwalteter Message Broker. IoT-Geräte können sich über einen sicheren Transportkanal mit dem Dienst verbinden und Daten austauschen. Diverse Mechanismen ermöglichen ein Routing der Nachrichten zu anderen AWS-Diensten. Ein entsprechendes Code Sample [▶ 325] demonstriert, wie Sie sich mit diesem Broker verbinden können.
AWS IoT Greengrass	AWS IoT Greengrass ist als Bestandteil von AWS IoT Core ein Service, welcher typischerweise in einer Edge Device Umgebung läuft. Durch den integrierten Message Broker wird eine MQTT Kommunikation mit unterlagerten Geräten ermöglicht. Ein entsprechendes Code Sample [▶ 325] demonstriert, wie Sie sich mit diesem Broker verbinden können.
AWS IoT Shadow	Der AWS IoT Shadow speichert Statusinformationen eines IoT-Geräts im Rahmen des AWS IoT Core Diensts. Das Code Sample für AWS IoT Core [▶ 325] kann als Grundlage für die Kommunikation mit dem AWS IoT Shadow verwendet werden.
Bosch IoT Suite	Die Bosch IoT Suite ist eine Produktfamilie in der Bosch IoT Cloud, welche eine Datenkommunikation und -analyse, sowie ein Device Management von IoT-Geräten ermöglicht. Durch eine MQTT-Schnittstelle wird der Empfang und Versand von Telemetriedaten ermöglicht. Ein entsprechendes Code Sample [▶ 330] demonstriert, wie Sie sich mit diesem Broker verbinden können.
CloudMQTT Broker	Message Broker als verwalteter Cloud-Dienst. Unser allgemeines Code Sample zum Herstellen einer MQTT Verbindung [▶ 319] kann als Grundlage für eine Datenkommunikation verwendet werden.
IBM Watson IoT	IBM Watson IoT ist eine Produktfamilie in der IBM Cloud, welche eine Datenkommunikation und -analyse, sowie ein Device Management von IoT-Geräten ermöglicht. Durch eine MQTT-Schnittstelle wird der Empfang und Versand von Telemetriedaten ermöglicht. Ein entsprechendes Code Sample [▶ 330] demonstriert, wie Sie sich mit diesem Broker verbinden können.
HiveMQ Message Broker	HiveMQ ist ein Message Broker, welcher einen schnellen und sicheren Datenaustausch über das MQTT Protokoll ermöglicht. Unser allgemeines Code Sample zum Herstellen einer MQTT Verbindung [▶ 319] kann als Grundlage für eine Datenkommunikation verwendet werden.

Cloud-Dienst oder Software	Beschreibung
Homebridge	Homebridge erlaubt die Anbindung von IoT Geräten an das Apple HomeKit, welche ansonsten keine eigene native Apple HomeKit Unterstützung haben. Für Homebridge gibt es ein MQTT-Client Plugin, über welches man eine Verbindung mit einem Message Broker herstellen kann. Unser allgemeines Code Sample zum Herstellen einer MQTT Verbindung [▶ 319] kann als Grundlage für eine Datenkommunikation mit Homebridge verwendet werden.
MathWorks ThingSpeak	ThingSpeak ist ein Cloud-basierter Dienst, welcher eine Datenaggregation, Visualisierung und Analyse von IoT-Daten ermöglicht. Ein entsprechendes Code Sample [▶ 331] demonstriert, wie Sie sich mit diesem Broker verbinden können.
Microsoft Azure IoT Edge	Microsoft Azure IoT Edge ist eine Softwareapplikation, welche typischerweise in einer Edge Device Umgebung läuft. Durch den integrierten Message Broker wird eine MQTT Kommunikation mit unterlagerten Geräten ermöglicht. Sie können unser Code Sample für den Azure IoT Hub [▶ 327] verwenden, um eine MQTT Verbindung mit dieser Applikation aufzubauen.
Microsoft Azure IoT Hub	Microsoft Azure IoT Hub ist ein verwalteter Nachrichten-Dienst in der Microsoft Azure Cloud, welcher einen sicheren Austausch von IoT-Daten ermöglicht. Ein entsprechendes Code Sample [▶ 327] demonstriert, wie Sie sich mit diesem Broker verbinden können.
Microsoft Azure IoT Hub Device Twin	Der Microsoft Azure IoT Hub Device Twin speichert Statusinformationen eines IoT-Geräts im Rahmen des Microsoft Azure IoT Hub. Ein entsprechendes Code Sample [▶ 333] demonstriert, wie Sie Daten mit dem Device Twin austauschen können.
Mosquitto Message Broker	Mosquitto ist ein Message Broker, welcher einen schnellen und sicheren Datenaustausch über das MQTT Protokoll ermöglicht. Unser allgemeines Code Sample zum Herstellen einer MQTT Verbindung [▶ 319] kann als Grundlage für eine Datenkommunikation verwendet werden.
Node-RED	Node-RED ist ein Programmierwerkzeug, welches eine grafische Programmierung erlaubt, um Geräte miteinander zu verbinden. Node-RED hat einen integrierten MQTT-Client, welcher sich zur Kommunikation mit einem Message Broker eignet und hierüber mit TwinCAT verbunden werden kann. Unser allgemeines Code Sample zum Herstellen einer MQTT Verbindung [▶ 319] kann als Grundlage für eine Datenkommunikation mit Node-RED verwendet werden.

Cloud-Dienst oder Software	Beschreibung
Shelly	Shelly Geräte bieten von Hause aus einen MQTT-Client, der sich mit einem Message Broker verbinden und darüber Daten mit anderen Geräten, z.B. TwinCAT, austauschen kann. Unser allgemeines Code Sample zum Herstellen einer MQTT Verbindung [▶ 319] kann als Grundlage für eine Datenkommunikation mit einem Shelly-Gerät verwendet werden.

4.4 Applikationsbeispiele

4.4.1 Senden von SMS- und E-Mail-Nachrichten

Viele Maschinenapplikationen verwenden SMS- und E-Mail-Benachrichtungen, um Statusinformationen und Alarme zu versenden. Diese Dokumentation beschreibt einen cloud-basierten Ansatz für das Versenden von SMS- und E-Mail-Nachrichten. Bei diesem Ansatz ist der Kommunikationskanal vom eigentlichen Nachrichtentyp (SMS, E-Mail) abstrahiert und die Entscheidung, ob eine eingehende Nachricht der Maschine als SMS oder E-Mail (oder beides) zu behandeln ist, wird in der Cloud getroffen.

Historie

Klassischerweise wurde für den Versand von SMS- oder E-Mail-Nachrichten entweder eine Telefoneinwahl (über ein angeschlossenes USB- oder serielles Modem) oder eine SMTP-Verbindung mit einem Mailserver direkt von der Maschinensteuerung verwendet. Auch wenn dies sehr gut funktioniert haben mag, sind die Nachteile dieses Ansatzes schnell klar:

- Es wird spezielle Modem-Hardware und ein Vertrag mit einem Mobilfunkanbieter benötigt, wodurch zusätzliche Kosten entstehen.
- Es wird ein weiterer Kommunikationskanal zu einem Mailserver benötigt.

TwinCAT-seitig unterstützt die TwinCAT3 Function TF6350 TC3 SMS/SMTP den SPS-Programmierer beim Versand von SMS- und E-Mail-Nachrichten. Um SMS-Nachrichten zu senden, kommuniziert das Supplementprodukt mit einem GSM-Modem, das über einen seriellen RS232-Anschluss mit der Steuerung verbunden ist. Für E-Mail-Nachrichten stellt das Supplementprodukt eine SMTP-Verbindung mit einem Mailserver im Netzwerk her. Danach ist der Mailserver für die Nachrichtenzustellung verantwortlich.

Voraussetzungen

Als Voraussetzung muss folgendes vorhanden sein, bevor Sie mit dieser Dokumentation fortfahren.

- Stellen Sie sicher, dass Sie ein AWS-Konto erstellt haben und dass Sie mit den Anmeldedaten dieses Kontos auf die AWS Management-Konsole zugreifen können.

> **Andere TwinCAT IoT-Produkte und -Protokolle**
>
> In diesem Tutorial werden wir TF6701 IoT Communication für die Verbindung mit AWS IoT Core über MQTT verwenden. Beachten Sie, dass andere Produkte aus der TwinCAT IoT-Produktreihe ebenfalls eine Verbindung mit AWS IoT Core herstellen können und dass auch für diese Produkte dieselben Grundsätze gelten. Beachten Sie außerdem, dass MQTT nur ein Transportkanal ist, der für die Verbindung mit AWS IoT Core verwendet werden kann. Eine weitere Transportoption ist HTTPS, das mit dem TwinCAT IoT-Produkt TF6760 IoT HTTPS/REST implementiert werden kann.

- Installieren Sie TwinCAT 3.1 Build 4022.0 oder höher, damit Ihnen das Produkt TF6701 IoT Communication zur Verfügung steht. Wir empfehlen, falls möglich, auf die neueste TwinCAT-Version zu wechseln.
- Um zu verstehen, wie TwinCAT mit AWS IoT Core verbunden wird, verweisen wir auf die TF6701-Dokumentation.

Wir empfehlen, beim Lesen dieser Dokumentation die folgenden Artikel über erste Schritte mit AWS zu lesen. Wir werden darauf hinweisen, wenn es wichtig ist, einen dieser Artikel zu lesen, bevor Sie mit dem nächsten Schritt fortfahren.

AWS IoT Core: https://docs.aws.amazon.com/iot/latest/developerguide/iot-gs.html

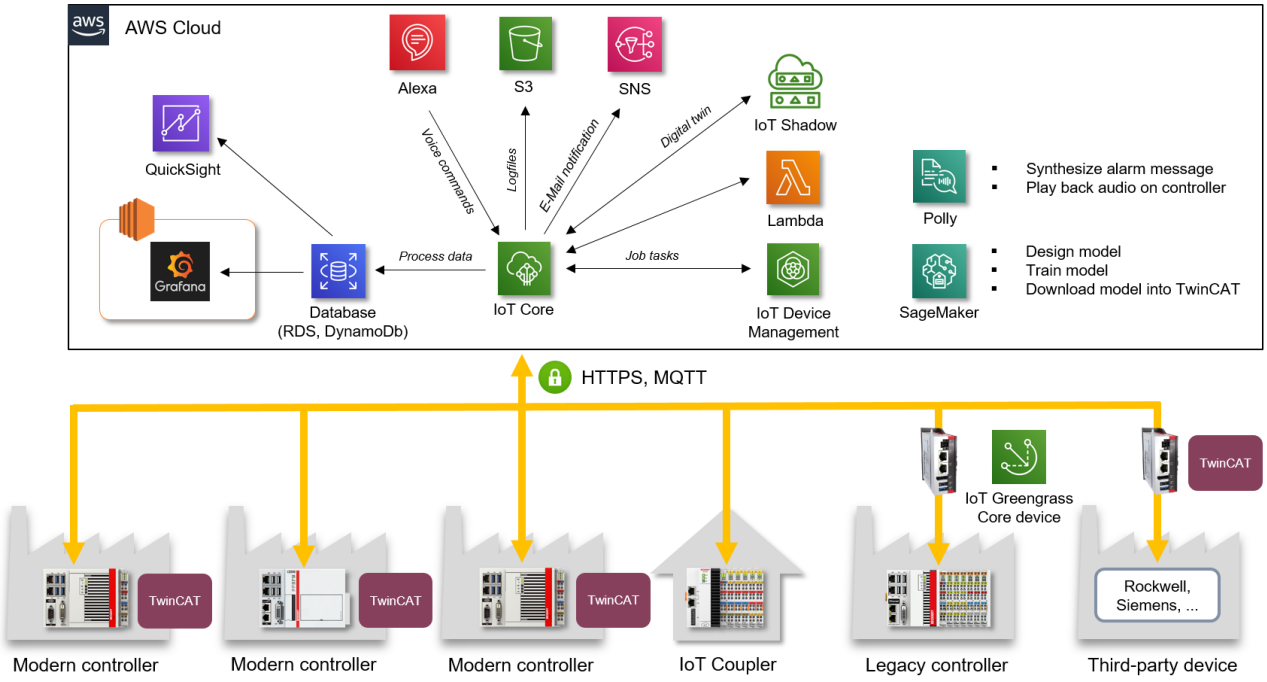
Amazon SNS: https://docs.aws.amazon.com/sns/latest/dg/sns-getting-started.html

Verwendung von Amazon SNS in einer AWS IoT Core-Regel: https://docs.aws.amazon.com/iot/latest/developerguide/config-and-test-rules.html

Architektur

Cloud-Systeme bieten die technische Infrastruktur, die das Internet of Things (IoT) ermöglicht, und sorgen für die erforderliche Skalierbarkeit, um Millionen verbundener Geräte zu unterstützen. Cloud-Serviceanbieter wie Microsoft Azure und Amazon Web Services (AWS) bieten Hunderte von Diensten an, um verschiedene Anwendungsfälle zu unterstützen: virtuelle Maschinen, Message-Broker, Datenbanken, serverlose Funktionen usw. Ihr Produktportfolio umfasst auch verschiedene Funktionen für die Verarbeitung von Nachrichten von verbundenen Geräten. Ein zentraler Message-Broker (manchmal auch als „IoT Hub“ bezeichnet) bietet einen einzigen, sicheren Endpunkt für die Kommunikation der Geräte mit anderen Diensten im Ecosystem. Mithilfe sogenannter „Regeln“ kann der Benutzer dann Nachrichten filtern und einfach an andere Dienste weiterleiten. Da der Message-Broker als einziger Endpunkt für alle Datenkonnektivitätsszenarios behandelt wird, wird die Firewall-Angriffsfläche minimiert.

Die folgende Grafik veranschaulicht dieses Konzept und basiert auf Diensten, die von AWS angeboten werden. Ähnliche Dienste finden sich auch auf anderen Cloud-Plattformen wie Microsoft Azure, AWS dient lediglich als Beispiel.



Reduzieren wir dies auf unseren Anwendungsfall (SMS- und E-Mail-Nachrichten), sind die betreffenden Dienste daher:

- AWS IoT Core (Message-Broker), um einen einzigen, sicheren Endpunkt in der Cloud zu haben
- Amazon SNS, um SMS- und E-Mail-Nachrichten zu senden

Die Hauptvorteile einer cloud-basierten Lösung gegenüber dem klassischen Setup sind folgende:

1. Der Nachrichtentyp (E-Mail, SMS, ...) ist transparent für das Gerät, das die Nachricht ausgibt. Das Gerät sendet einfach seine Nachricht und die Entscheidung, ob die Nachricht eine E-Mail oder eine SMS sein wird, wird in der Cloud getroffen.
2. Adressänderungen müssen nicht an das Gerät weitergeleitet werden, das eine Nachricht ausgibt. Alle Adressen (E-Mail-Adressen, Telefonnummern, ...) werden in der Cloud verwaltet.
3. Sicherer Transport zwischen dem Gerät, das eine Nachricht ausgibt, und der Cloud. Jede Nachricht des Geräts wird über einen sicheren Kommunikationskanal an AWS IoT Core gesendet – entweder über MQTT oder HTTPS.

4. Die erforderliche Konnektivität mit dem Internet basiert auf TCP/IP und es wird keine zusätzliche Modem-Hardware zum Senden von SMS benötigt.
5. Ein Vertrag mit einem Mobilfunkanbieter ist nicht erforderlich. AWS verwaltet den Transport einer SMS und berechnet Ihnen dies nutzungsabhängig (Pay-per-Use).

TwinCAT IoT unterstützt MQTT- und HTTPS-Konnektivität mit AWS IoT Core. Auf den folgenden Seiten der Dokumentation wird ausführlicher beschrieben, wie die an diesem Anwendungsfall beteiligten Komponenten konfiguriert werden.

Einrichtung von AWS IoT Core

AWS IoT Core ist ein skalierbarer und verwalteter Message-Broker-Dienst im AWS-Ecosystem. Er ermöglicht es, Geräte sicher zu verbinden und ihren Data Ingest zu verwalten. Um AWS IoT Core zu verwenden, ist Folgendes erforderlich:

- Ein AWS-Konto, um sich bei der webbasierten AWS Management-Konsole anzumelden. Alle benötigten AWS IoT Core-Funktionen werden automatisch implementiert, wenn das Konto erstellt wird, so dass eine Servicebereitstellung nicht notwendig ist.
- Geräteanmeldedaten (Zertifikate) und Sicherheitsrichtlinien für jedes verbundene Gerät. Die Zertifikate müssen auf das Gerät übertragen und während der Verbindungsherstellung mit dem AWS IoT Core-Dienst von ihm verwendet werden. Anders ausgedrückt: die Zertifikate werden von TwinCAT IoT immer dann verwendet, wenn die Verbindung aufgebaut wird. Dies wird in einem späteren Kapitel beschrieben.

Die Einrichtung von AWS IoT Core umfasst die folgenden Topics:

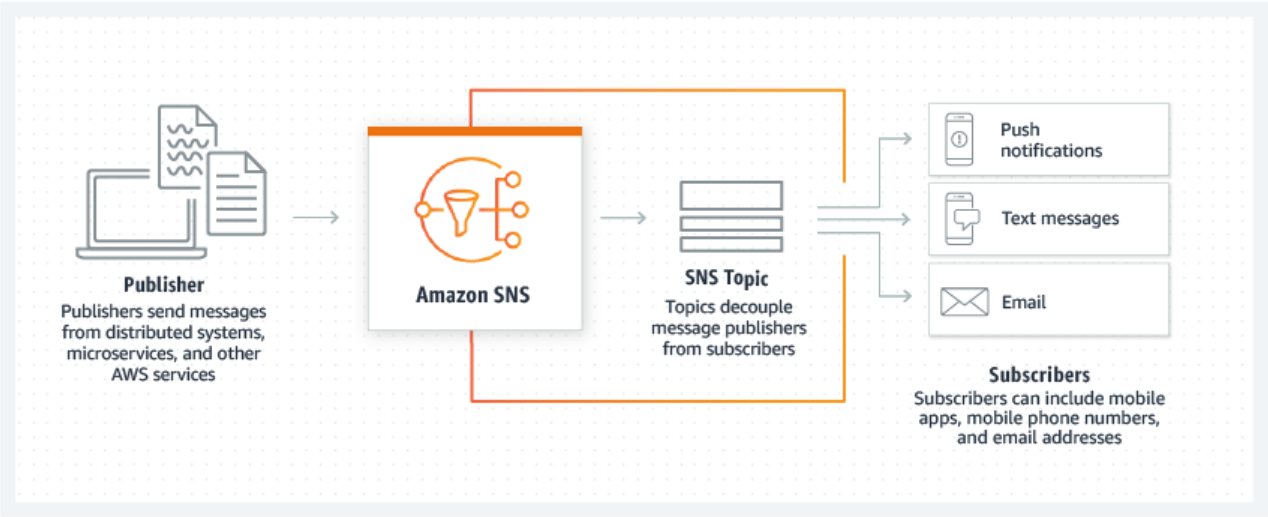
1. Anmelden an der AWS IoT-Konsole
2. Erstellen eines Objekts
3. Registrieren eines Geräts
4. Konfigurieren Ihres Geräts
5. Anzeigen der MQTT-Nachrichten des Geräts mit dem AWS IoT MQTT-Client
6. Konfigurieren und Testen von Regeln (nächstes Kapitel)

Diese Schritte werden im Tutorial Erste Schritte mit AWS IoT Core ausführlich beschrieben. Dieses Tutorial ist eine gute Dokumentationsquelle und enthält ausführliche Informationen über die oben genannten Schritte. Wir empfehlen, dieses Tutorial zu lesen und die schrittweise Anleitung durchzuarbeiten, bevor Sie mit dem nächsten Kapitel in dieser Dokumentation fortfahren.

Schritt 6 des Tutorials beschreibt genau den Anwendungsfall, den wir zu lösen versuchen: Konfigurieren einer AWS IoT Core-Regel, die Amazon SNS verwendet, um eine E-Mail- oder SMS-Nachricht zu senden.

Einrichtung von Amazon SNS

Amazon SNS ermöglicht es Ihnen, Push-Benachrichtigungen an mobile Apps, Textnachrichten an Mobilfunknummern und reine Text-E-Mails an E-Mail-Adressen zu senden. In Schritt 6 („Konfigurieren und Testen von Regeln“) der offiziellen Anleitung „Erste Schritte mit AWS IoT Core“ wird ausführlich beschrieben, wie Amazon SNS vorbereitet werden muss, damit eine AWS IoT-Regel eine Gerätenachricht an ein sogenanntes SNS Topic weiterleiten kann. Wenn Sie Ihr Wissen über Amazon SNS vertiefen möchten, empfehlen wir das Tutorial Erste Schritte mit Amazon SNS.

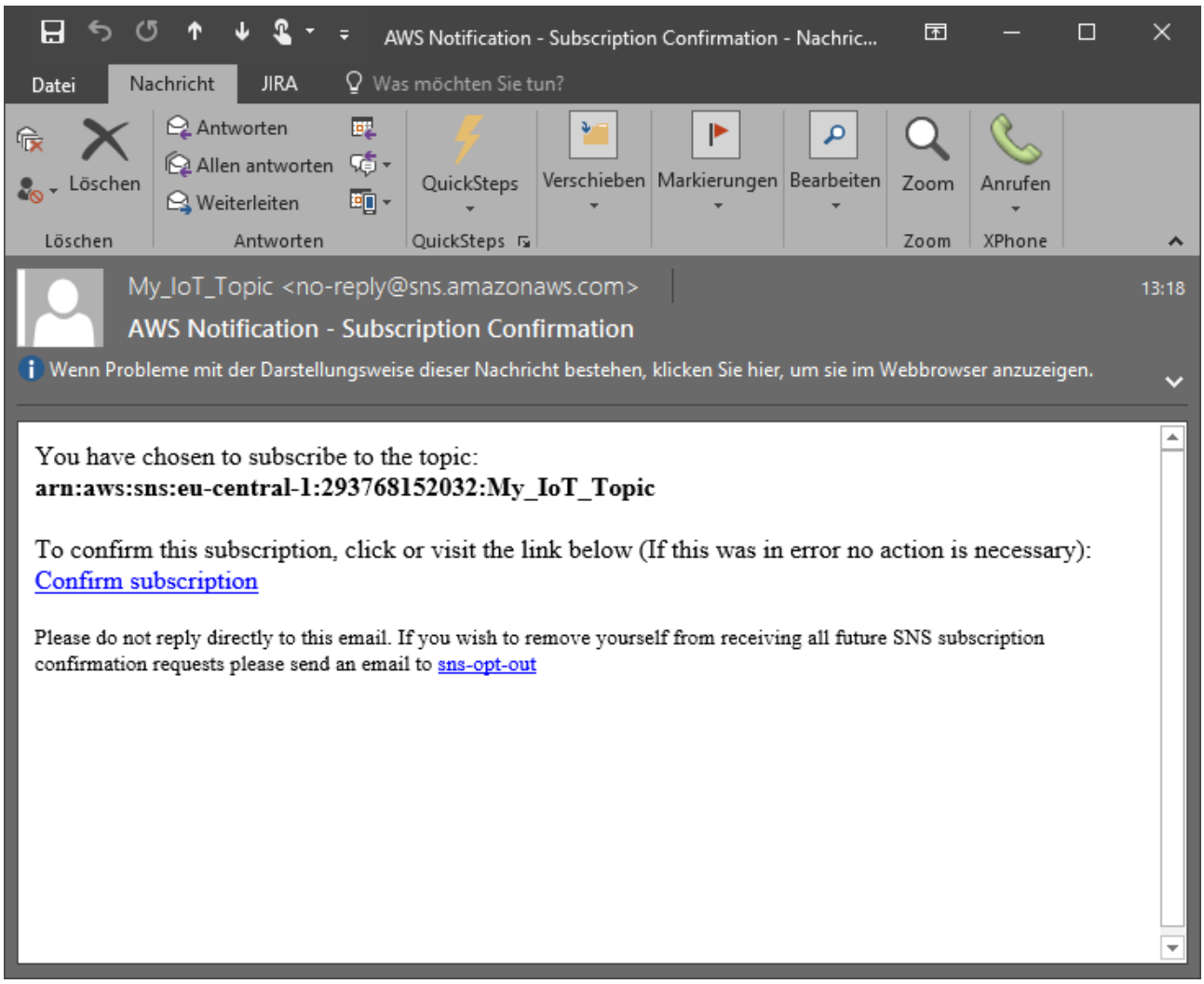


In dieser offiziellen Grafik fungiert der AWS IoT Core als Zwischenstation zwischen dem Publisher einer Nachricht (dem „Gerät“) und Amazon SNS.

Folgen Sie einfach den in der offiziellen Anleitung „Erste Schritte“ beschriebenen Schritten, um

- ein Amazon SNS Topic und ein Abonnement zu erstellen (E-Mail als „Protokoll“ verwenden)
- eine AWS IoT Core-Regel zu erstellen
- die Regel zu testen

Beachten Sie folgendes: Bei Verwendung von „E-Mail“ als Protokoll erhält die eingegebene E-Mail-Adresse eine erste, einmalige E-Mail zur Abonnementbestätigung. Diese E-Mail muss bestätigt werden, bevor Nachrichten an diese E-Mail-Adresse gesendet werden können. Dies ist mit Newslettern vergleichbar.



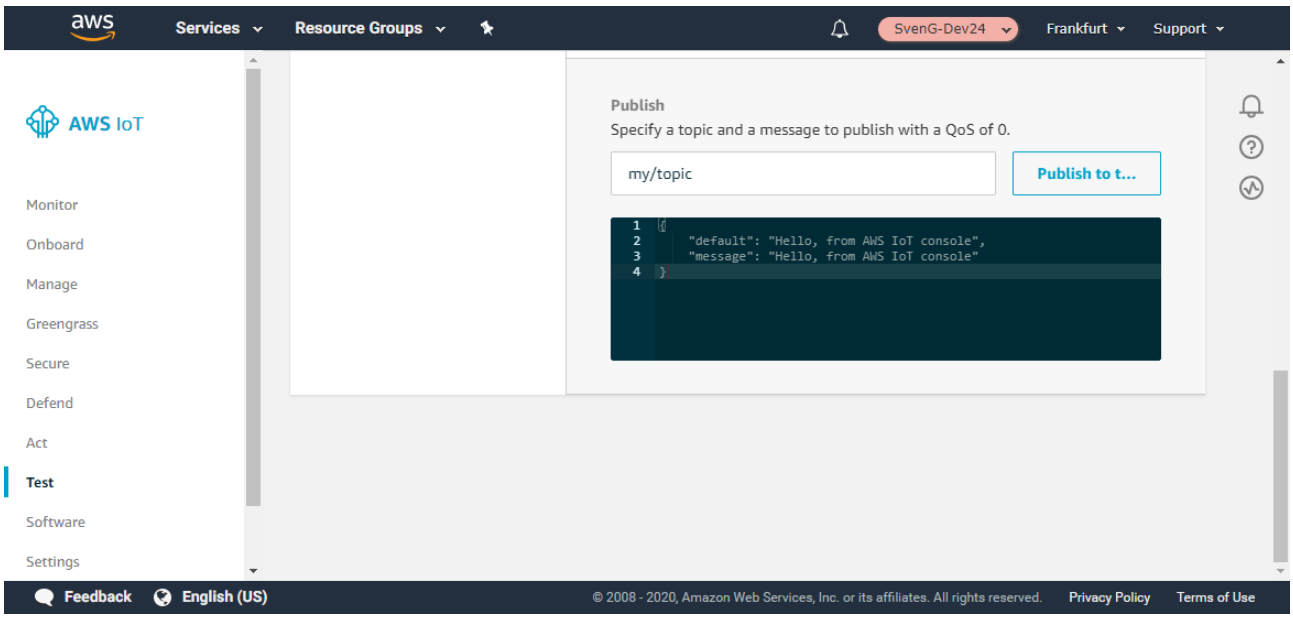
Servicelimits

Beachten Sie, dass es bei regulären AWS-Konten eine Grenze für SMS-Nachrichten geben kann. Diese Grenze kann erhöht werden, indem Sie den AWS-Support aufrufen. Weitere Informationen finden Sie auf der Webseite Amazon SNS Servicelimit.

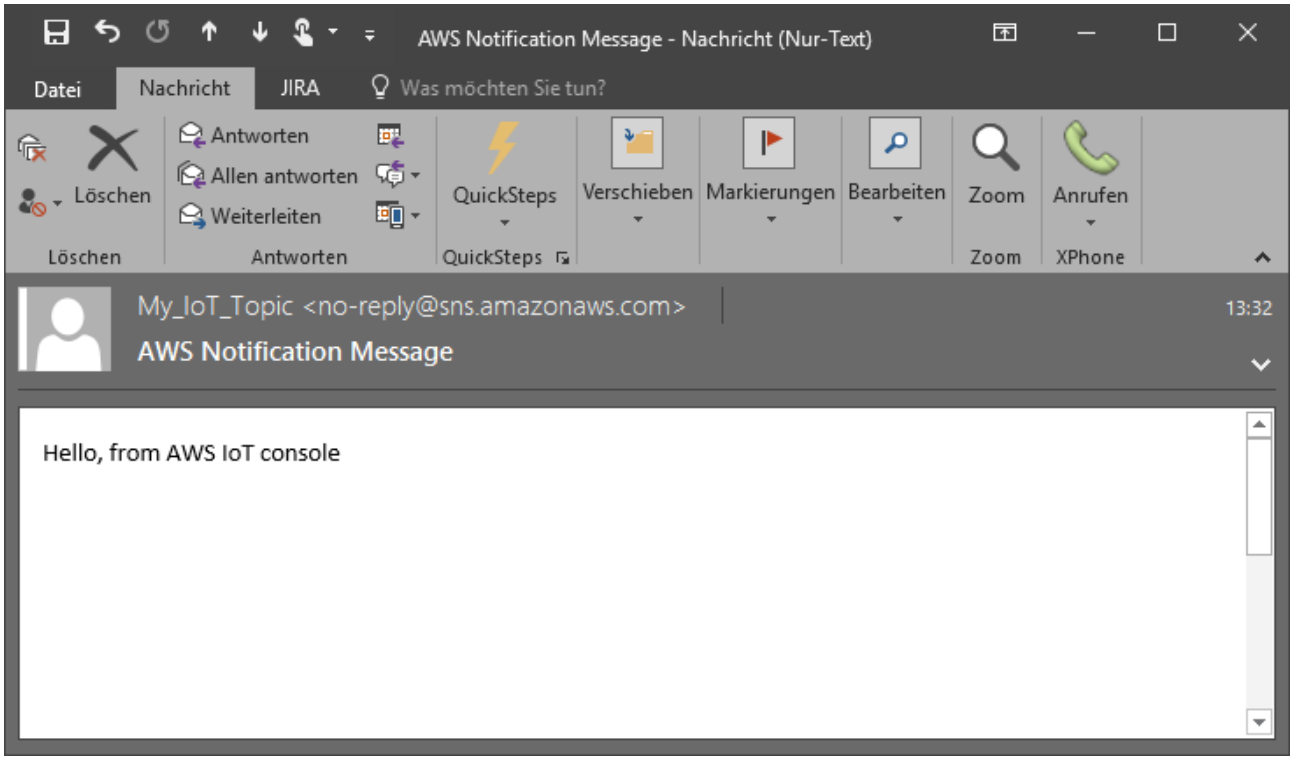
Nachdem sowohl das Amazon SNS Topic/Subscription als auch die AWS IoT-Regel erstellt wurden, kann das Setup mit Hilfe des MQTT-Clients, der in die AWS Management-Konsole integriert ist, getestet werden. Senden Sie dazu lediglich eine Testnachricht an das Topic „my/topic“, das in der AWS IoT-Regel als Filter festgelegt worden ist, und verwenden Sie den folgenden JSON-Inhalt:

```
{
  "default": "Hello, from AWS IoT console",
  "email": "Hello, from AWS IoT console"
}
```

Alle verfügbaren Eigenschaften sind im Rahmen der Amazon SNS-Dokumentation dokumentiert.



Nach einigen Sekunden sollte die Benachrichtigungs-E-Mail bei der E-Mail-Adresse eingehen, die für die Amazon SNS-Subscription verwendet worden ist.



Im nächsten Schritt möchten wir es TwinCAT ermöglichen, Nachrichten an AWS IoT Core zu senden.

Einrichtung von TwinCAT

Die TwinCAT IoT Supplement-Produkte ermöglichen die Cloud-Konnektivität für verschiedene Anwendungsfälle. Einer ihrer Hauptvorteile liegt darin, dass sie Standardkommunikationsprotokolle verwenden, um die Konnektivität mit Cloud-Systemen verschiedener Anbieter, z. B. Microsoft Azure, Amazon Web Services, IBM, Google usw., zu ermöglichen.

In dieser Dokumentation möchten wir mit Hilfe von TF6701 IoT Communication eine Verbindung mit AWS IoT Core herstellen und eine Nachricht für das Topic „my/topic“ veröffentlichen, das in der AWS IoT-Regel als Filter festgelegt worden ist, um die Nachrichten weiterzuleiten, die für dieses bestimmte Topic bei Amazon SNS eingehen, um eine Benachrichtigungs-E-Mail zu senden.

Voraussetzungen

Dieses Kapitel basiert auf dem regulären TF6701-Beispiel IotMqttSampleAwsIoT, das zeigt, wie die Verbindung mit AWS IoT Core im Allgemeinen hergestellt wird. Laden Sie dieses Beispiel herunter, damit eine gemeinsame Ausgangsbasis vorhanden ist. Sehen Sie sich auch die entsprechende Infosys-Website für dieses bestimmte Beispiel an, um ausführlichere Informationen über die Funktionsweise des Beispielcodes zu finden.

Wichtig

Bevor Sie fortfahren, müssen alle im Kapitel Einrichtung von AWS IoT Core dieser Dokumentation beschriebenen Schritte ausgeführt worden sein.

Verbindungsherstellung

Alle Zertifikate, die mit der AWS Management-Konsole erstellt worden sind, müssen in der Datenstruktur FB_IotMqttClient.stTLS referenziert sein (sCA, sCert und sKeyFile). Verwenden Sie die URL der AWS IoT Core-Instanz als sHostName, wie auf der AWS Management-Konsole gezeigt wird. Da die Verbindung eine sichere MQTT-Verbindung ist, verwenden Sie 8883 als nHostPort. Die MQTT-Client-ID (sClientId) steht für den Objektnamen (ThingName), der bei der Erstellung des Objekts im Kapitel Einrichtung von AWS IoT Core verwendet wurde.

```
(* TLS settings for AWS IoT connection *)
fbMqttClient.stTLS.sCA := 'c:\certs\AmazonRootCA1.pem';
fbMqttClient.stTLS.sCert := 'c:\certs\6a1ba937cb-certificate.pem.crt';
fbMqttClient.stTLS.sKeyFile := 'c:\certs\6a1ba937cb-private.pem.key';

(* Broker settings für AWS IoT *)
fbMqttClient.sHostName:= 'aXX-ats.iot.eu-central-1.amazonaws.com';
fbMqttClient.nHostPort:= 8883;
fbMqttClient.sClientId:= 'ThingName';
```

Festlegen der richtigen Topics

Das Standardbeispiel veranschaulicht, wie die Verbindung mit AWS IoT Core hergestellt wird, um Daten mit diesem Message-Broker auszutauschen. Es veröffentlicht Nachrichten für ein Topic und abonniert ein Topic, um Nachrichten zu erhalten. In dem Beispiel sind beide Themen gleich, so dass TwinCAT dieselbe Nachricht erhält, die es an den Broker gesendet hat.

Zusätzlich zu diesem regulären Beispielverhalten werden wir nun neuen Code schreiben, damit das Beispiel eine Nachricht für das Topic „my/topic“ sendet, so dass eine E-Mail-Nachricht gesendet wird. Hierzu deklarieren wir zunächst einige neue Variablen:

```
sTopicEmail : STRING(255) := 'my/topic';
bSendEmail : BOOL;
sPayloadEmail : STRING(255) := '{"default": "Hello from TwinCAT","message": "Hello from TwinCAT"}';
```

Dann werden wir nach der IF-Abfrage für die Timer-Ausführung die folgenden Codezeilen hinzufügen:

```
IF fbTimer.Q THEN
  ...
END_IF
IF bSendEmail THEN
  bSendEmail := FALSE;
  fbMqttClient.Publish(sTopic:= sTopicEmail, pPayload:= ADR(sPayloadEmail), nPayloadSize:=
LEN2(ADR(sPayloadEmail)), eQoS:= TcIotMqttQos.AtMostOnceDelivery, bRetain:= FALSE, bQueue:= FALSE);
END_IF
```

Nach der Aktivierung des Projekts validieren Sie zuerst, ob die Verbindung mit AWS IoT Core erfolgreich war, indem Sie den Parameter eConnectionState prüfen (muss stabil „MQTT_ERR_SUCCESS“ sein). Wenn der Verbindungsstatus zu „wechseln“ scheint oder einen TLS-Fehler anzeigt (z. B. „MQTT_ERR_TLS_VERIFICATIONFAILED“), überprüfen Sie die Schritte des Kapitels „Einrichtung von AWS IoT Core“ nochmals und kontrollieren Sie, ob das Gerätezertifikat aktiviert worden ist und die Sicherheitsrichtlinie dem Gerät erlaubt, Daten für die verwendeten Topics zu veröffentlichen.

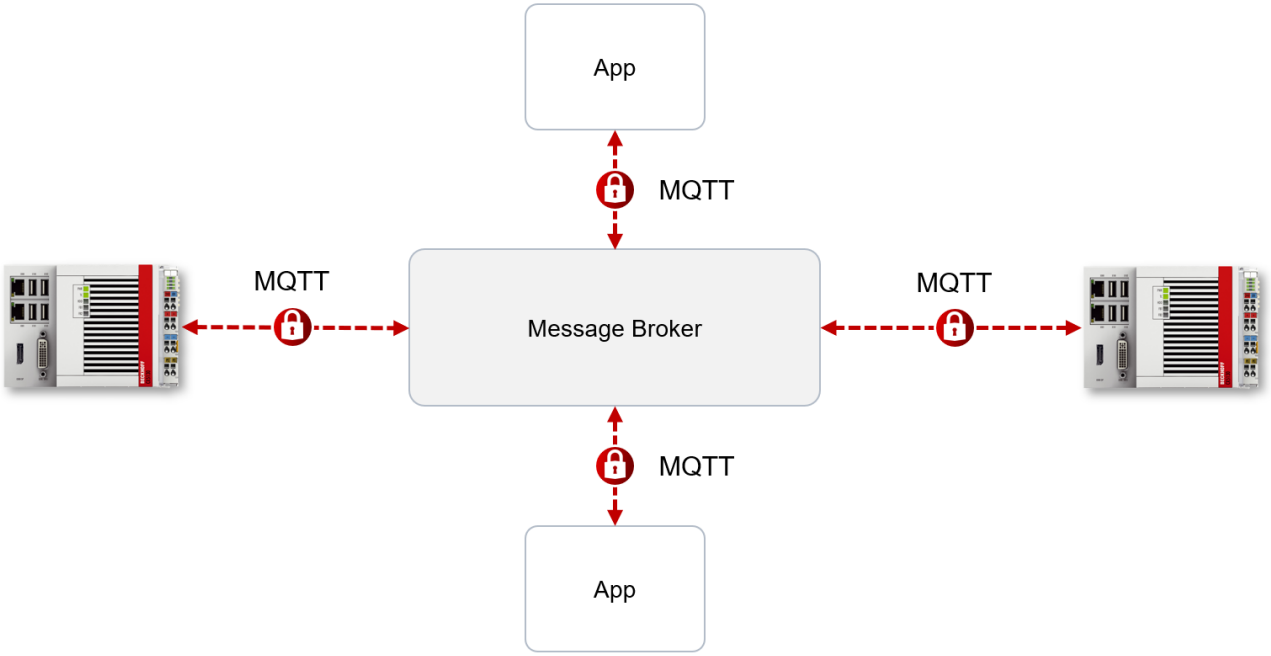
Wenn die Verbindung erfolgreich ist, versuchen Sie, bSendEmail auf TRUE zu setzen. Nach einigen Sekunden sollte eine E-Mail im Posteingang der E-Mail-Adresse erscheinen, die für die Amazon SNS Subscription verwendet worden ist.

4.5 Protokollgrundlagen

4.5.1 Überblick

MQTT (Message Queueing Telemetry Transport) ist ein Publisher/Subscriber-basiertes Kommunikationsprotokoll, welches eine Nachrichten-basierte Übertragung zwischen Applikationen ermöglicht. Eine zentrale Komponente bei dieser Art der Übertragung ist der sogenannte Message Broker, bei welchem es sich somit um eine Nachrichten-orientierte Middleware handelt.

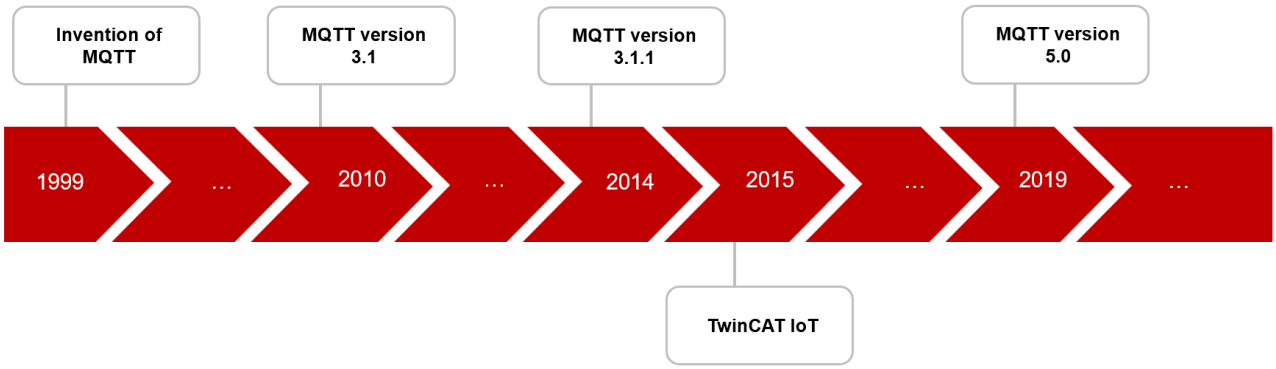
Der Message Broker hat die Aufgabe, Nachrichten zwischen den einzelnen Applikationen, bzw. dem Sender und Empfänger einer Nachricht, zu verteilen. Er entkoppelt dabei Sender und Empfänger voneinander, sodass diese keine gegenseitigen Addressinformationen kennen und austauschen müssen. Alle Kommunikationsteilnehmer wenden sich beim Senden und Empfangen an den Message Broker und dieser übernimmt die Verteilung der Nachrichten.



Bei der Verbindungsherstellung eines Clients zum Message Broker können Sicherheitsmechanismen wie TLS (Transport Layer Security) oder auch Benutzername-/Passwort-Authentifizierung eingesetzt werden, um die Kommunikationsverbindung zu verschlüsseln und eine Authentifizierung zwischen Client und Message Broker zu realisieren.

4.5.2 Protokoll-Versionen

Das MQTT Protokoll bzw. dessen Spezifikation gibt es bereits seit Ende der 1990er Jahre. Im Jahr 2019 wurde die Version 5.0 der Spezifikation freigegeben, welche das Protokoll um eine Vielzahl an neuen Eigenschaften bereichert.



4.5.3 ClientID

Beim Herstellen einer Verbindung mit dem Message Broker übermittelt der Client eine sogenannte ClientID, welche zur eindeutigen Identifizierung des Clients auf dem Message Broker dient. Der MQTT-Kommunikationstreiber aus TwinCAT3 erzeugt automatisch eine eigene ClientID, welche sich an dem folgenden Namensschema orientiert:

PlcProjectName-TcMqttClient%n

%n ist hierbei ein inkrementeller Zähler für die Nummer der jeweiligen MQTT Client Instanz. Jede Instanz des Funktionsbausteins FB_IotMqttClient erhöht hierbei diesen Zähler. In den meisten Fällen ist das Verwenden dieses ClientID Formats ausreichend. In speziellen Fällen, z.B. abhängig vom Message Broker oder auch durch die eigene MQTT Applikation bedingt, muss eine anwendungspezifische ClientID vergeben werden. Dies kann über einen entsprechenden Eingang an den Funktionsbausteinen FB_IotMqttClient [▶ 57] und FB_IotMqtt5Client [▶ 81] erfolgen.

Soll eine eindeutige ClientID automatisch beim Start des SPS Projekts generiert werden, so bietet sich die Verwendung einer GUID an, welche über den Funktionsbaustein FB_CreateGuid aus der Bibliothek Tc2_System erzeugt werden kann. Der folgende Beispielcode verdeutlicht die Verwendung dieses Funktionsbausteins.

```
PROGRAM MAIN
VAR
  fbGuid : FB_CreateGUID;
  objGuid : GUID;
  sGuid : STRING;
  nState : UINT;
  bStart : BOOL; // set to TRUE to start this sample
END_VAR

CASE nState OF
  0 :
    IF bStart THEN
      bStart := FALSE;
      nState := nState + 1;
    END_IF

  1 : // create GUID using FB_CreateGuid from Tc2_System library
    fbGuid(bExecute := TRUE, pGuidBuffer := ADR(objGuid), nGuidBufferSize := SIZEOF(objGuid));
    IF NOT fbGuid.bBusy THEN
      fbGuid(bExecute := FALSE);
      IF NOT fbGuid.bError THEN
        nState := nState + 1;
      ELSE
        nState := 255; // go to error state
      END_IF
    END_IF

  2: // GUID has been created, now convert to STRING
    sGuid := GUID_TO_STRING(objGuid);
    nState := nState + 1;

  3: // done

255: // error state

END_CASE
```

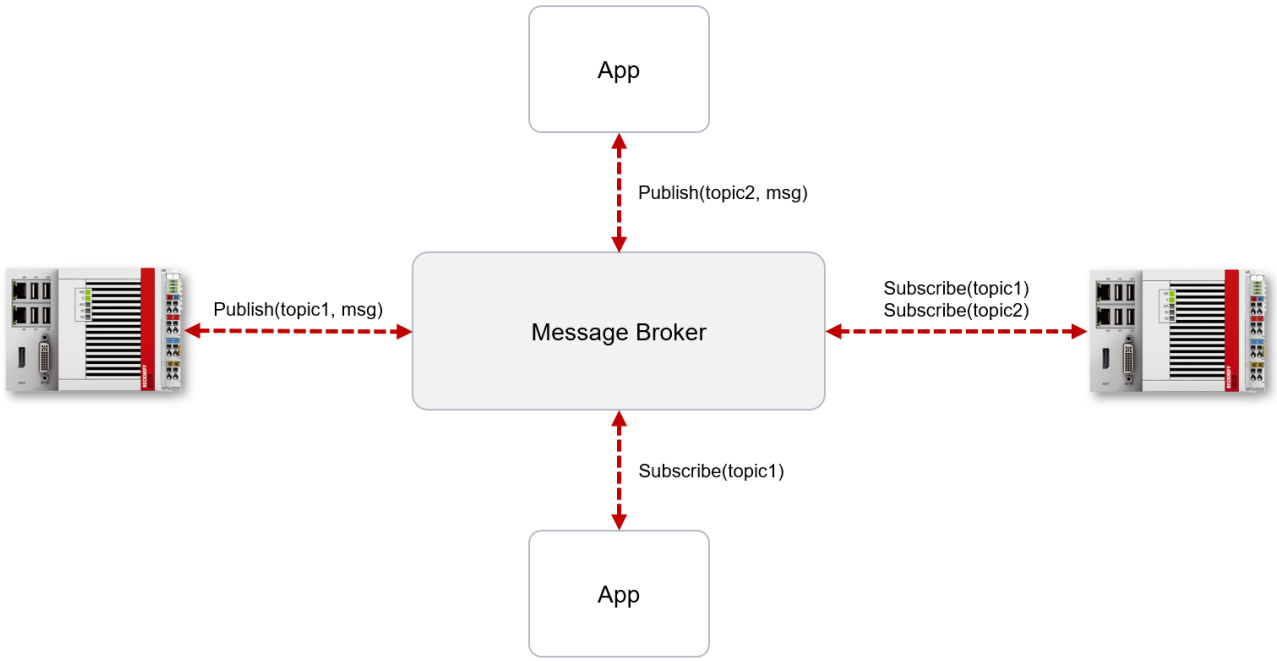
Nach Ausführung dieser State Machine enthält die Variable sGuid die generierte GUID als STRING. Diese kann dann an den Funktionsbausteinen FB_IotMqttClient und FB_IotMqtt5Client als ClientID verwendet werden.

4.5.4 Topics

Der Message Broker verwaltet die Adressierung von Nachrichten über sogenannte „Topics“. Ein Topic kann man sich als ein hierarchisch organisiertes Postfach vorstellen, bei dem die einzelnen Hierarchieebenen durch ein „/“ voneinander getrennt sind. Beispiel:

```
Campus/Building1/Floor2/Room3/Temperature
```

Der Publisher einer Nachricht gibt beim Versand immer an, für welches Topic eine Nachricht gedacht ist. Ein Subscriber hingegen gibt an, für welches Topic er sich interessiert. Der Message Broker leitet dann die Nachrichten entsprechend weiter.



Wildcards

Bei der Verwendung von Topics können auch sogenannte „Wildcards“ benutzt werden. Eine Wildcard ersetzt einen Teil des Topics. Ein Subscriber erhält dann ggf. Nachrichten aus mehreren Topics. Es werden zwei Arten von Wildcards unterschieden:

- Single Level Wildcards
- Multi Level Wildcards

Beispiel Single Level Wildcard:

Das +-Symbol beschreibt eine Single Level Wildcard. Wird es z.B. vom Subscriber wie folgt verwendet, so werden entsprechende Nachrichten an die Topics entweder vom Subscriber empfangen oder nicht empfangen.

- Der Empfänger subscribed sich auf Campus/Building1/Floor2/**+**/Temperature
- Der Publisher sendet an Campus/Building1/Floor2/Room1/Temperature - OK
- Der Publisher sendet an Campus/Building1/Floor2/Room2/Temperature - OK
- Der Publisher sendet an Campus/Building42/Floor1/Room1/Temperature - NOK
- Der Publisher sendet an Campus/Building1/Floor2/Room1/Fridge/Temperature - NOK

Beispiel Multi Level Wildcard:

Das #-Symbol beschreibt eine Multi Level Wildcard. Wird es z.B. vom Subscriber wie folgt verwendet, so werden entsprechende Nachrichten an die Topics entweder vom Subscriber empfangen oder nicht empfangen. Das #-Symbol muss hierbei immer als letztes Symbol im Topic-String verwendet werden.

- Der Empfänger subscribed sich auf Campus/Building1/Floor2/**#**
- Der Publisher sendet an Campus/Building1/Floor2/Room1/Temperature - OK
- Der Publisher sendet an Campus/Building1/Floor2/Room2/Temperature - OK
- Der Publisher sendet an Campus/Building42/Floor1/Room1/Temperature - NOK
- Der Publisher sendet an Campus/Building1/Floor2/Room1/Fridge/Temperature – OK
- Der Publisher sendet an Campus/Building1/Floor2/Room1/Humidity - OK

4.5.5 Datenformate

MQTT erlaubt den Transport von Binärdaten, wobei der eigentliche Aufbau des Nachrichteninhalts nicht durch die Spezifikation festgeschrieben wird. Es können somit beliebige Daten transportiert werden, z.B. zyklische oder event-basierte Telemetriedaten oder auch Kommandos an die Steuerung. Da das Datenformat nicht spezifiziert ist, muss das Format dem Sender und Empfänger bekannt sein.

Über die Jahre sind hierbei eine Vielzahl an Datenformaten und entsprechenden -beschreibungssprachen entstanden. In vielen IoT-Anwendungen haben sich JSON und XML als Beschreibungssprachen etabliert. Der Aufbau eines entsprechenden JSON- oder XML-Dokuments ist jedoch weiterhin applikationsspezifisch und muss den Anwendungen bekannt sein, damit diese Daten miteinander austauschen können.

JSON

Bei JSON (JavaScript Objection Notation) handelt es sich um eine schlanke Beschreibungssprache in einer einfach lesbaren Textform, in der Daten über Property/Value-Paare in sogenannten Objekten organisiert werden. JSON hat sich auf Grund der einfachen Lesbarkeit und des (im Vergleich mit anderen Beschreibungssprachen wie z.B. XML) immer noch geringen Overheads in den meisten IoT-Anwendungen etabliert. Das folgende Beispiel zeigt ein JSON-Dokument, welches Telemetriewerte von drei Sensoren inklusive Metadaten und Zeitstempel beinhaltet.

```
{
  "Timestamp": "2017-04-04T12:42:42",
  "Values": {
    "Sensor1": 42.41999816894531,
    "Sensor2": 230,
    "Sensor3": 3
  },
  "MetaData": {
    "Sensor1": {
      "Unit": "m/s",
      "DisplayName": "Speed"
    },
    "Sensor2": {
      "Unit": "V",
      "DisplayName": "Voltage"
    },
    "Sensor3": {
      "Unit": "A",
      "DisplayName": "Current"
    }
  }
}
```

XML

Bei XML (Extensible Markup Language) handelt es sich um eine Beschreibungssprache zur Strukturierung von Daten, um diese sowohl für den Menschen als auch für ein Programm leserlich aufzubereiten. XML hat einen deutlich größeren Overhead als JSON und findet in modernen IoT-Applikationen kaum noch Anwendung. Das oben gezeigte Beispiel ließe sich wie folgt exemplarisch in XML darstellen.

```
<XmlDocument>
  <Timestamp>2017-04-04T12:42:42</Timestamp>
  <Values>
    <Value Name="Sensor1">42.41999816894531</Value>
    <Value Name="Sensor2">230</Value>
    <Value Name="Sensor3">3</Value>
  </Values>
  <MetaDataColl>
    <MetaData Name="Sensor1">
      <Unit>m/s</Unit>
      <DisplayName>Speed</DisplayName>
    </MetaData>
    <MetaData Name="Sensor2">
      <Unit>V</Unit>
      <DisplayName>Voltage</DisplayName>
    </MetaData>
    <MetaData Name="Sensor3">
      <Unit>A</Unit>
      <DisplayName>Current</DisplayName>
    </MetaData>
  </MetaDataColl>
</XmlDocument>
```

Natürlich gibt es noch viele weitere Varianten, wie sich das oben genannte JSON in ein äquivalentes XML überführen lässt (und hierbei wird auch deutlich warum dennoch Sender und Empfänger das Datenformat kennen müssen – trotz JSON oder XML). Es soll hierbei nur das grundlegende Prinzip aufgezeigt werden.

Vergleicht man die beiden oben stehenden Dokumente, so fällt schnell der Größenunterschied auf: während das JSON-Dokument eine ungefähre Größe (abhängig von etwaigen Zeilenumbrüchen und Leerzeichen) von circa 393 Bytes aufweist, so hat das äquivalente XML-Dokument eine Größe von circa 569 Bytes und ist somit um circa 44% größer (obwohl die gleichen Informationen übertragen werden).

SPS-Bibliothek Tc3_JsonXml

Zur einfachen Erstellung und Verarbeitung sowohl von JSON- als auch von XML-Objekten gibt es die Bibliothek Tc3_JsonXml, welche automatisch mit TwinCAT 3 XAE installiert wird. (Siehe Dokumentation PLC Lib: Tc3_JsonXml)

4.5.6 QoS

QoS (Quality-of-Service) ist eine Vereinbarung zwischen dem Client und Message Broker in Bezug auf das Garantieren der Nachrichtenübermittlung. Das sogenannte QoS-Level muss hierbei sowohl beim Publish- als auch beim Subscribe-Vorgang vom Client angegeben werden. Es existieren drei verschiedene QoS-Level:

- 0 – Nachricht wird höchstens einmal gesendet
- 1 – Nachricht wird mindestens einmal gesendet
- 2 – Nachricht wird genau einmal gesendet

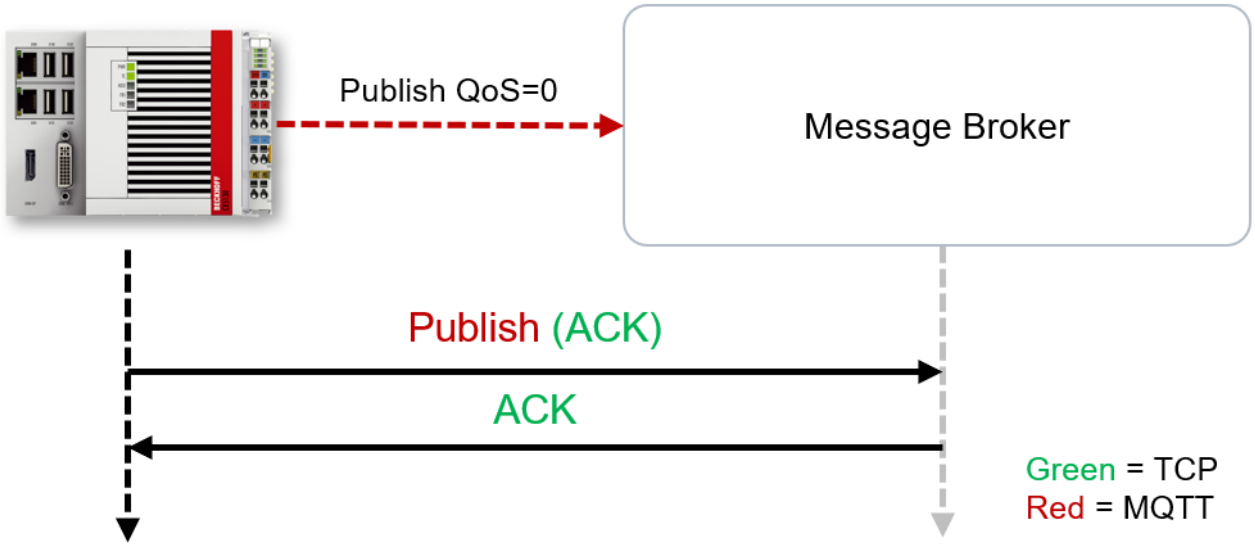
Zusammenspiel von Publisher- und Subscriber-QoS

Die MQTT-Spezifikation (MQTT5, Kapitel 3.8.4) definiert ein paar grundlegende Regeln beim Zusammenspiel von Publisher- und Subscriber-QoS, insbesondere wenn Publisher und Subscriber ein unterschiedliches Qos-Level angeben.

QoS-Level 0

Bei diesem QoS-Level erfolgt keine Bestätigung des Empfängers, ob die Nachrichten empfangen wurden oder nicht. In der Folge wird die Nachricht auch kein zweites Mal gesendet.

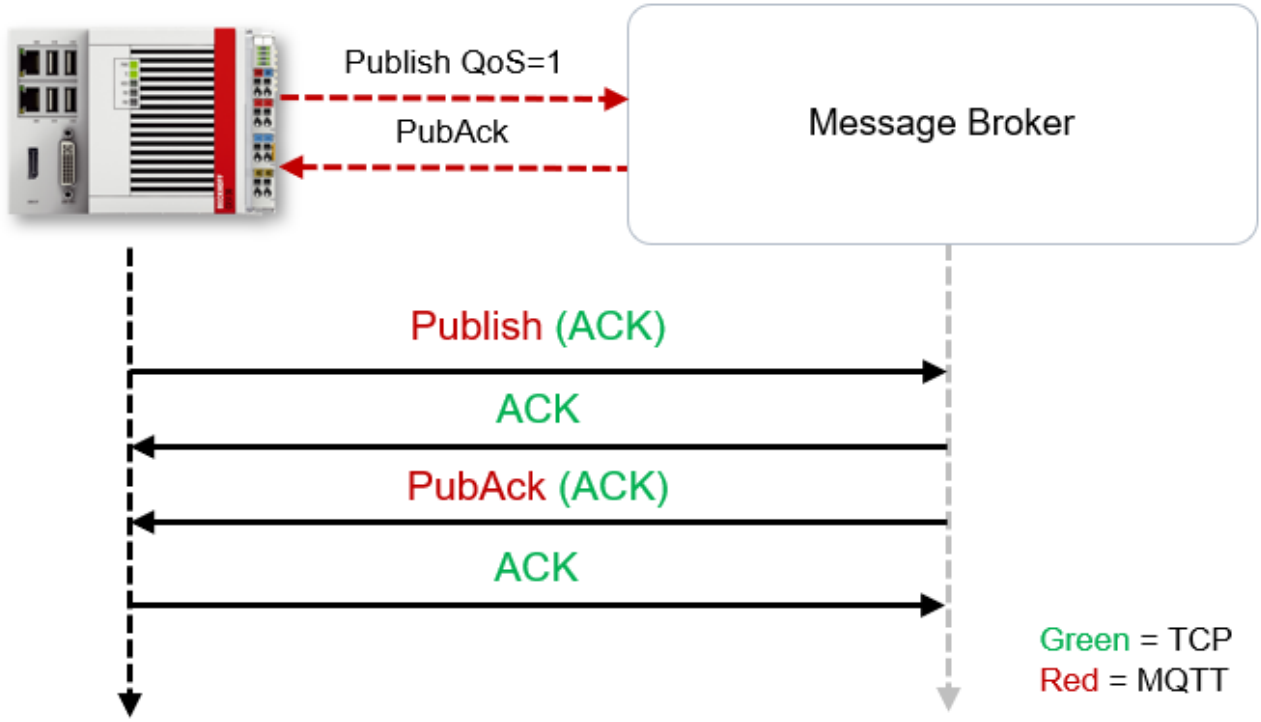
Die folgende Grafik veranschaulicht diesen Zusammenhang noch einmal, vor allem im Vergleich mit der unterlagerten TCP-Ebene.



QoS-Level 1

Bei diesem QoS-Level wird garantiert, dass die Nachricht zumindest einmal beim Empfänger ankommt. Aber die Nachricht kann unter Umständen auch mehrfach beim Empfänger eintreffen. Der Sender speichert die Nachricht intern bis er eine Bestätigung in Form einer PUBACK-Nachricht vom Empfänger erhält. Wenn die PUBACK-Nachricht für eine bestimmte Zeit ausbleibt, wird die Nachricht erneut gesendet.

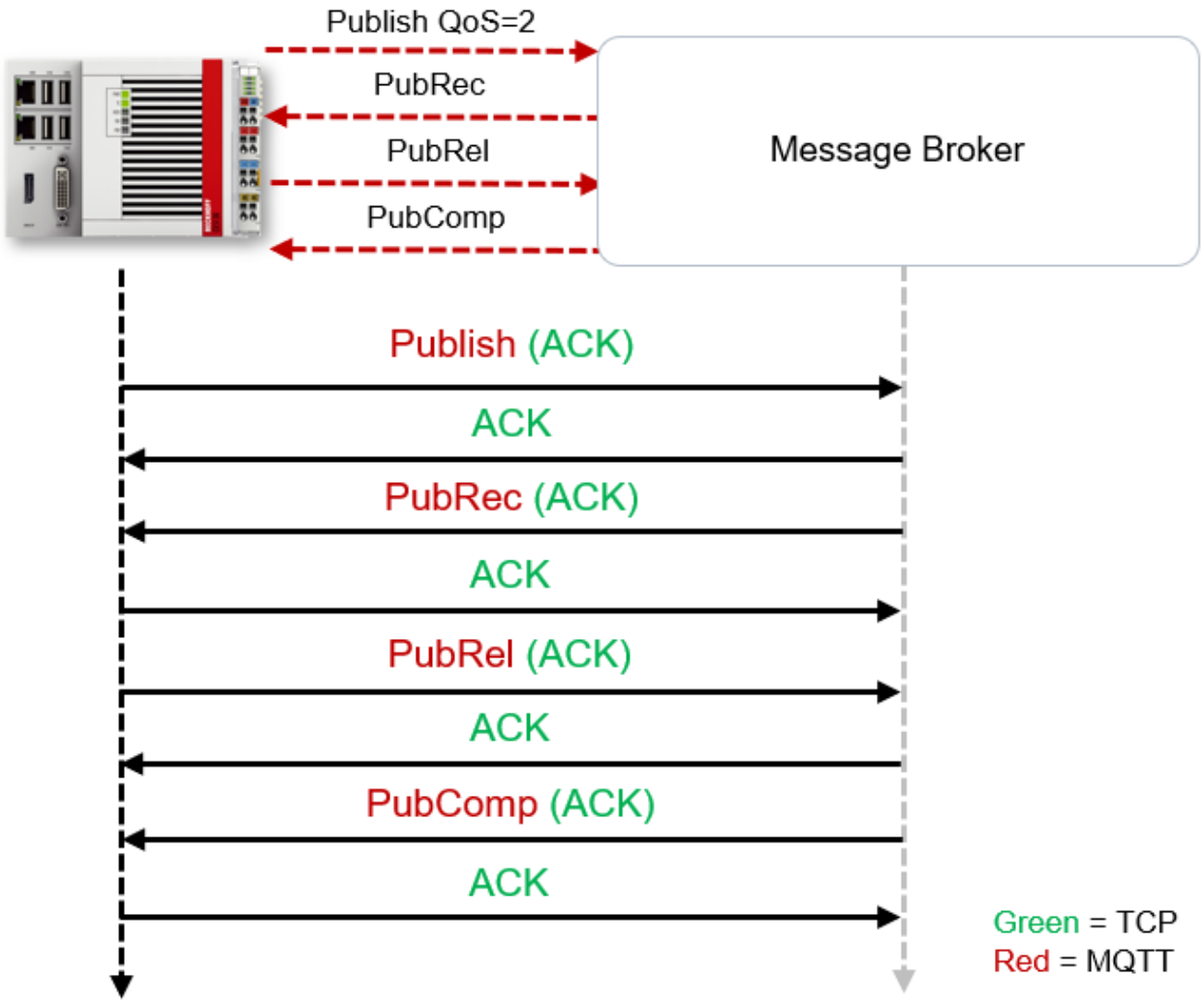
Die folgende Grafik veranschaulicht diesen Zusammenhang noch einmal, vor allem im Vergleich mit der unterlagerten TCP-Ebene.



QoS-Level 2

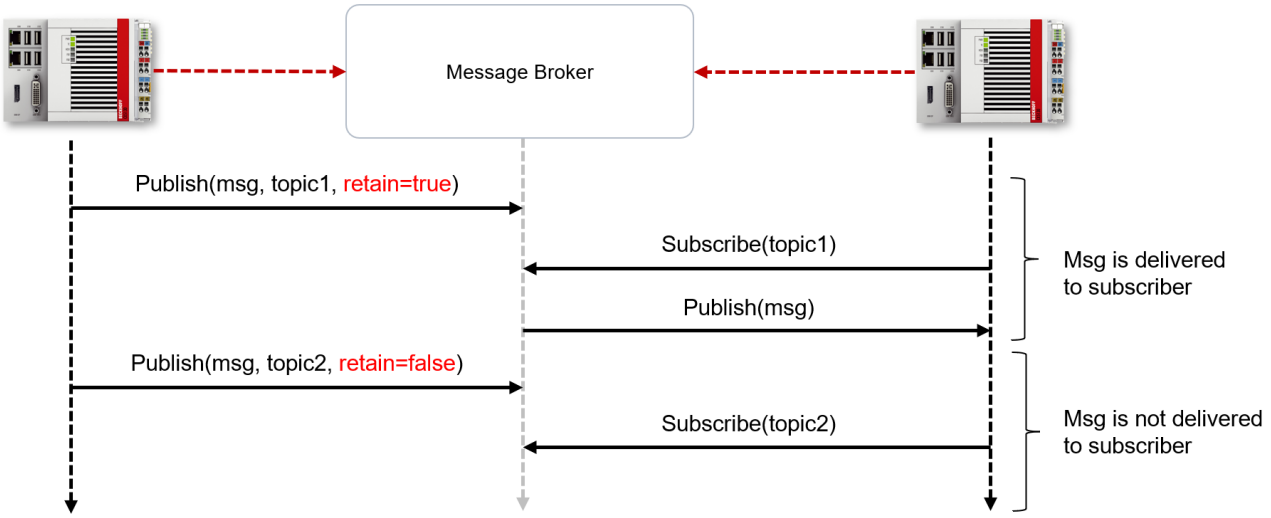
Bei diesem QoS-Level wird garantiert, dass die Nachricht maximal einmal beim Empfänger ankommt. Dies wird MQTT-seitig durch einen Handshake-Mechanismus realisiert. QoS-Level 2 ist der sicherste (aus Sicht der Nachrichtenübermittlung), aber auch langsamste QoS-Level. Wenn ein Empfänger eine Nachricht mit QoS-Level 2 erhält, bestätigt er die Nachricht mit einem PUBREC. Der Absender der Nachricht merkt sich diese intern bis er ein PUBCOMP empfangen hat. Dieser zusätzliche Handshake (verglichen mit QoS 1) ist wichtig, damit die Nachricht nicht doppelt übertragen wird. Wenn der Absender der Nachricht ein PUBREC erhält, kann er den initialen Publish verwerfen, da er weiß, dass die Nachricht einmal vom Empfänger empfangen wurde. Er merkt sich somit intern den PUBREC und sendet seinerseits ein PUBREL. Nachdem der Empfänger ein PUBREL empfangen hat, kann er die sich zuvor gemerkten Zustände verwerfen und mit einem PUBCOMP antworten. Umgekehrt genauso. Immer dann, wenn ein Paket verloren geht, ist der jeweilige Kommunikationsteilnehmer dafür verantwortlich, die zuletzt gesendete Nachricht nach einer bestimmten Zeit noch einmal zu senden.

Die folgende Grafik veranschaulicht diesen Zusammenhang noch einmal, vor allem im Vergleich mit der unterlagerten TCP-Ebene.



4.5.7 Retain

MQTT definiert keinen Mechanismus, damit dem Publisher einer Nachricht garantiert wird, dass ein Subscriber die Nachricht auch wirklich empfangen hat. Der Publisher kann über das QoS [▶ 31]-Level lediglich sicherstellen, dass die Nachricht sicher am Message Broker abgeliefert wurde. Auf der anderen Seite kann auch ein Subscriber nicht sicherstellen, wann ein Publisher eine Nachricht gesendet hat. Im worst-case subscribed sich der Empfänger erst auf das Topic, nachdem der Publisher eine Nachricht gesendet hat. In diesem Fall würde der Subscriber die Nachricht nicht empfangen. Ist das Retain-Flag beim Publish gesetzt, so würde der Subscriber es hingegen empfangen. Das folgende Schaubild verdeutlicht diesen Zusammenhang noch einmal:



Das sogenannte Retain-Flag bietet beim Publish-Vorgang die Möglichkeit, dem Message Broker mitzuteilen, dass dieser die Nachricht persistent in dem entsprechenden Topic vorhält. Der Message Broker speichert hierbei immer die zuletzt gesendete Retain-Nachricht. Jeder Client, der sich auf dieses Topic subscribed (entweder vor oder nach dem Publish) bekommt diese Nachricht dann sofort zugestellt sobald er die Subscription eingerichtet wird. Dieser Vorgang funktioniert ebenfalls für Wildcard Subscriptions [▶ 29].

Eine Retain-Nachricht kann auch wieder aus einem Topic entfernt werden, indem eine Nachricht ohne Payload bei gesetztem Retain-Flag an das entsprechende Topic gesendet wird.

4.5.8 Last will

Der LastWill ist eine Nachricht, die im Falle eines irregulären Verbindungsabbruches vom Broker an alle Clients gesendet wird, die das passende Topic abonniert haben. Verliert der MQTT-Client in der SPS die Verbindung zum Broker und es wurde beim Verbindungsaufbau ein LastWill hinterlegt, so wird dieser LastWill vom Broker kommuniziert, ohne dass der Client sich darum kümmern muss.

Bei einem geplanten Disconnect wird der LastWill laut Spezifikation nicht zwingend übertragen. Aus Sicht des SPS-Programmierers kann dieser entscheiden, ob er vor Aufruf des Disconnects den LastWill publishen will. Dazu wird auf dem LastWill-Topic die LastWill-Nachricht noch einmal gepublished. Das ist notwendig, da der Broker aufgrund der regulären Verbindungsabbruches die Last Will-Nachricht nicht veröffentlichen würde.

Bei einem TwinCAT-Kontextwechsel und einem daraus folgenden Neustart der MQTT-Kommunikation sendet der IoT-Treiber den vorher spezifizierten LastWill an den Broker, weil in diesem Moment aus der SPS keine Möglichkeit mehr dazu besteht. Wenn bei Verbindungsherstellung kein LastWill definiert wurde, wird auch keine Nachricht vor dem Disconnect übertragen.

4.5.9 Erweiterungen MQTT5

4.5.9.1 Request/Response

Während bei MQTT3 ein Request/Response-Verfahren noch manuell durch entsprechende, applikative Handshake-Mechanismen der MQTT-Clients umgesetzt werden musste, bietet MQTT5 nun eine native Implementierung hierfür an. Das Verfahren basiert auf der grundlegenden Annahme, dass der Absender eines Requests ein sogenanntes „Response-Topic“ definiert und als Bestandteil der Nachricht mitschickt. Der Empfänger des Requests kann dieses Response-Topic dann verwenden, um seine Antwort zu verschicken. Das folgende Bild veranschaulicht diesen Vorgang.

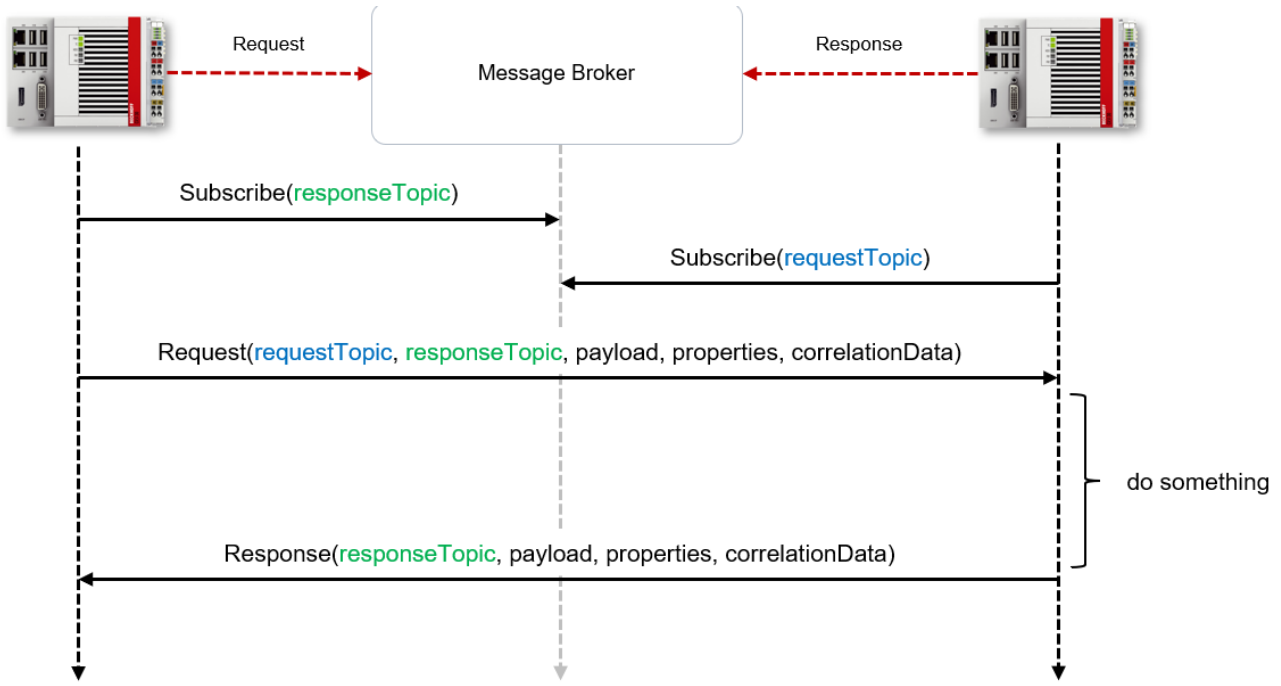


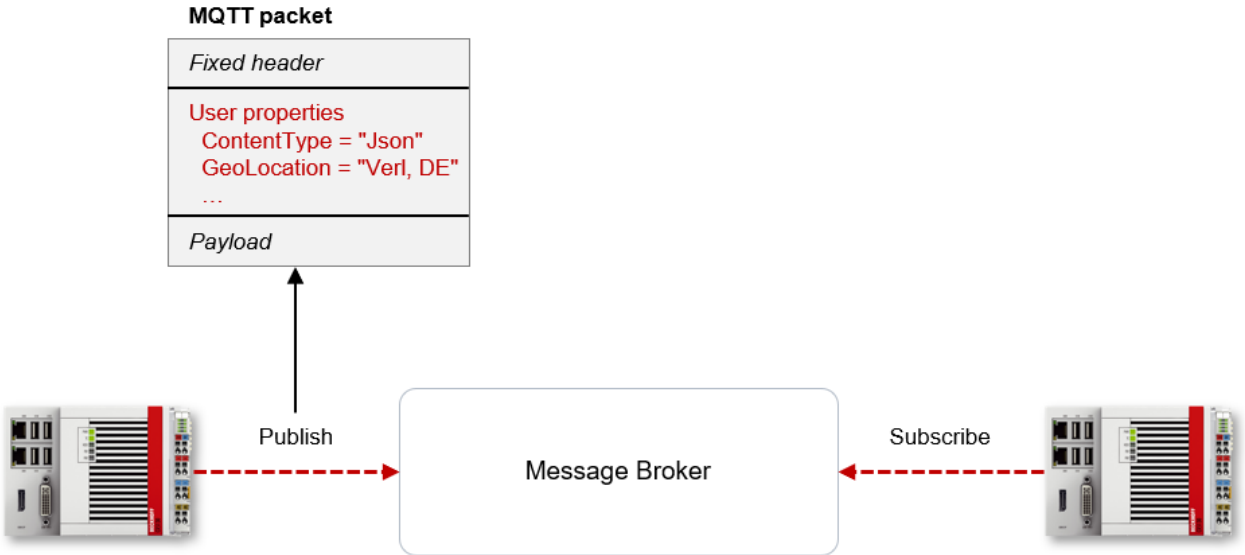
Abb. 1:

4.5.9.2 User properties

Eine der wichtigsten Neuerungen in MQTT5 sind die sogenannten User Properties. Hierbei handelt es sich um Key/Value Paare, welche bei nahezu allen MQTT Pakettypen gesetzt werden und zusätzliche Metadaten transportieren können – unabhängig vom eigentlichen Nachrichteninhalt. Es gibt keinerlei Begrenzung bei der Anzahl an User Properties, solange die Maximalgröße einer MQTT Nachricht nicht überschritten wird. Dieses Feature ist ähnlich zu den Key/Value Paaren, wie sie zum Beispiel in einem HTTP Header vorkommen.

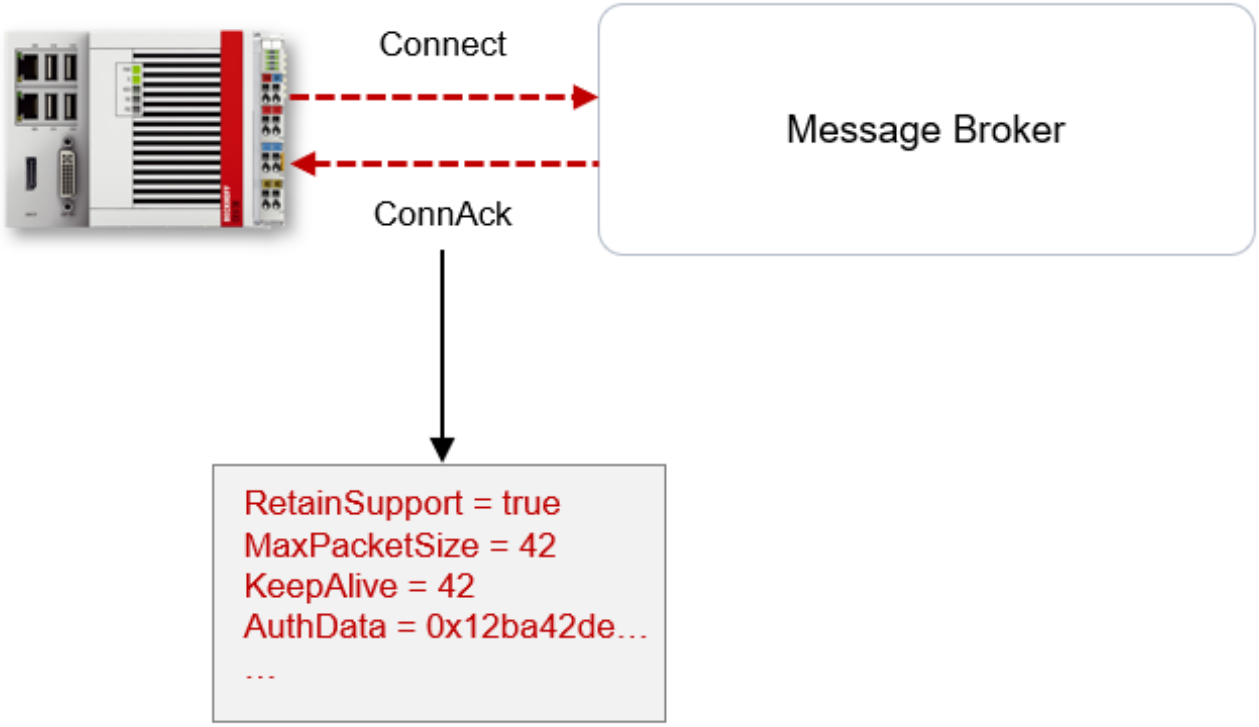
Der Vorteil von User Properties liegt in der Entkopplung vom eigentlichen Nachrichteninhalt. So können zum Beispiel Beschreibungsinformationen für den Nachrichteninhalt übermittelt werden, was das Encoding/ Decoding des Nachrichteninhalts auf Empfängerseite vereinfacht.

Des Weiteren können mit Hilfe von User Properties auch Routing-Informationen übermittelt werden, sodass einem Empfänger zum Beispiel mitgeteilt wird was er mit dem empfangenen Nachrichteninhalt tun soll.

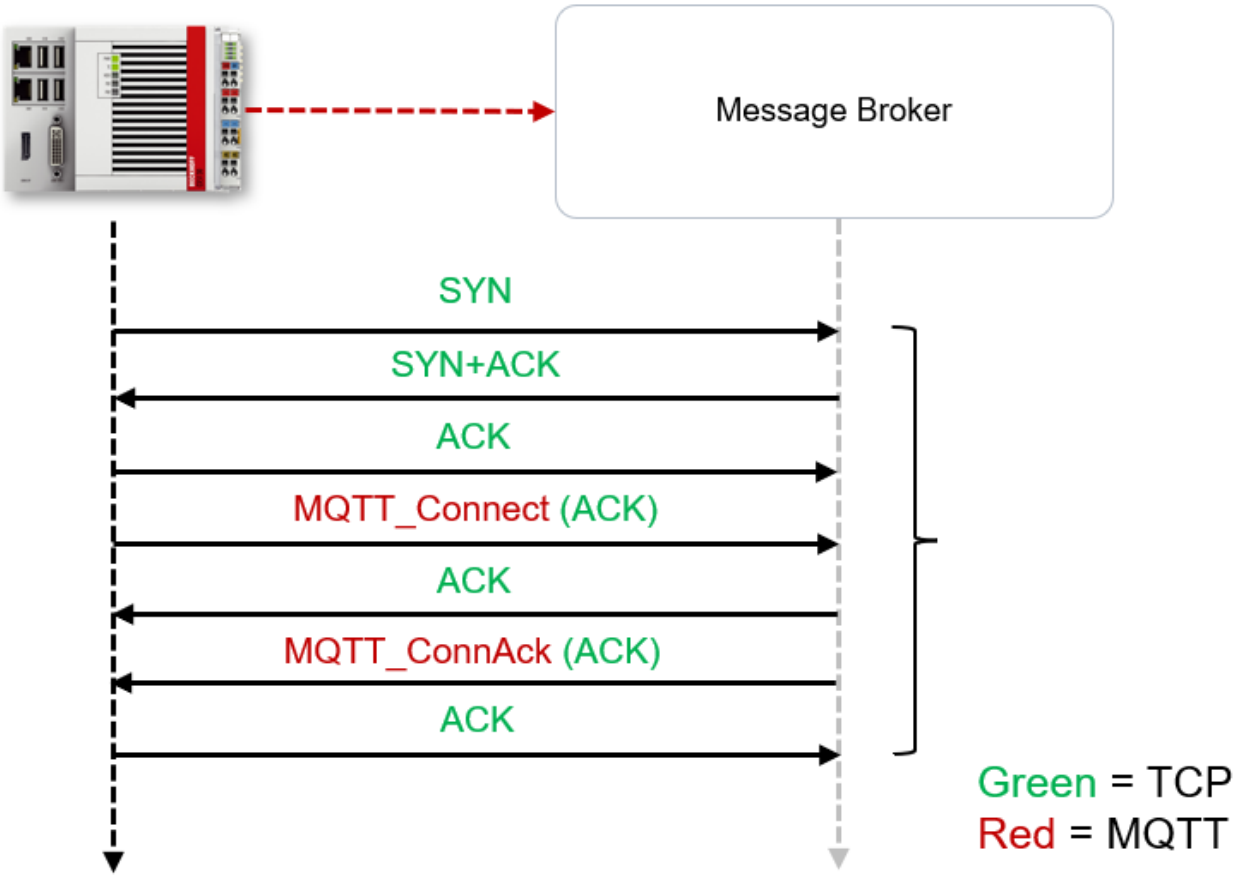


4.5.9.3 Connection acknowledgement

Die neu eingeführten ConnAck Return Codes ermöglichen es einem Message Broker dem Client anzuzeigen welche MQTT-Funktionen er unterstützt. Diese Information wird beim ConnAck Vorgang vom Message Broker an den Client übermittelt. Das folgende Bild veranschaulicht diesen Vorgang in vereinfachter Darstellung.



Im folgenden Bild wird der Verbindungsaufbau zwischen Client und Message Broker auf TCP-Ebene dargestellt:

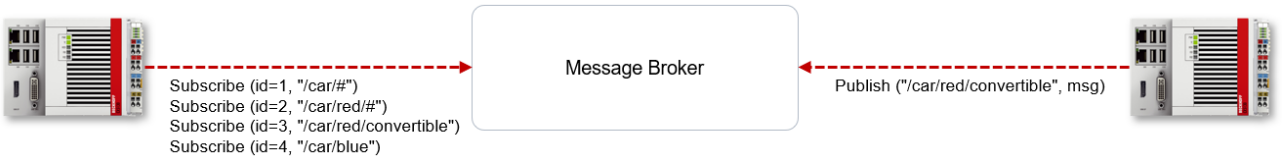


Die folgenden Informationen können beim Connection Acknowledgement vom Message Broker an den Client übermittelt werden.

Eigenschaft	Beschreibung
Retain available	Gibt an, ob der Message Broker Retain-Nachrichten unterstützt.
Server KeepAlive	Gibt die vom Server zugewiesene KeepAlive-Zeit an.
Shared subscriptions available	Gibt an, ob der Message Broker Shared Subscriptions unterstützt.
Wildcard subscriptions available	Gibt an, ob der Message Broker Wildcard Subscriptions unterstützt, z.B. Subscriptions auf ein # oder + Topic.
Authentication data	Beinhaltet die Authentifizierungsdaten, abhängig von der verwendeten Authentifizierungsmethode.
Maximum packet size	Gibt die maximale Paketgröße des Control Pakets an. Wenn dieser Wert nicht angegeben ist, dann ist keine Maximalgröße explizit gesetzt.
Maximum QoS	Gibt den maximalen QoS Level an, den der Message Broker unterstützt. Einige Message Broker Implementierungen unterstützen zum Beispiel kein QoS 2.
Reason code	Optionaler Reason Code, welchen der Message Broker im Rahmen des ConnAck Verfahrens an den Client übermitteln kann. Im Zusammenspiel mit der Server Reference (s.u.) kann der Message Broker dem Client hierüber z.B. mitteilen, dass er temporär oder permanent nicht erreichbar und/oder unter einer neuen Adresse erreichbar ist.
Maximum receive	Der Server verwendet diese Eigenschaft, um die Anzahl an QoS1 und QoS2 Publishes zu limitieren, die er für den Client bereit ist gleichzeitig zu verarbeiten.
Session expiry interval	Der Server kann hierüber den Client informieren, dass er ein anderes Session Expiry Interval verwendet als ursprünglich vom Client angefordert.
Response info	Optionale Rückgabeinformationen vom Message Broker an den Client. Ein Anwendungsfall hierfür kann zum Beispiel sein, dass der Broker dem Client einen „Topic-Bereich“ mitteilt der dem Client zugewiesen wurde und auf den er zugreifen kann.
Maximum topic alias	Gibt den maximalen Wert an, der für einen Topic Alias verwendet werden kann.
Assigned client ID	Gibt die vom Message Broker zugewiesene ClientID zurück, sofern der Client keine eigene ClientID angegeben hat.
Server reference	Optionale (Adress-) Referenz auf einen weiteren Message Broker, z.B. bei Reason Code 0x9C oder 0x9D (Server unavailable, Server has moved).
User Properties	User Properties sind Key/Value Paare, welche an die PublishProperties angehängt werden können. Die Bedeutung der UserProperties ist nicht Bestandteil der MQTT5 Spezifikation und somit applikationsspezifisch.

4.5.9.4 Subscription identifier

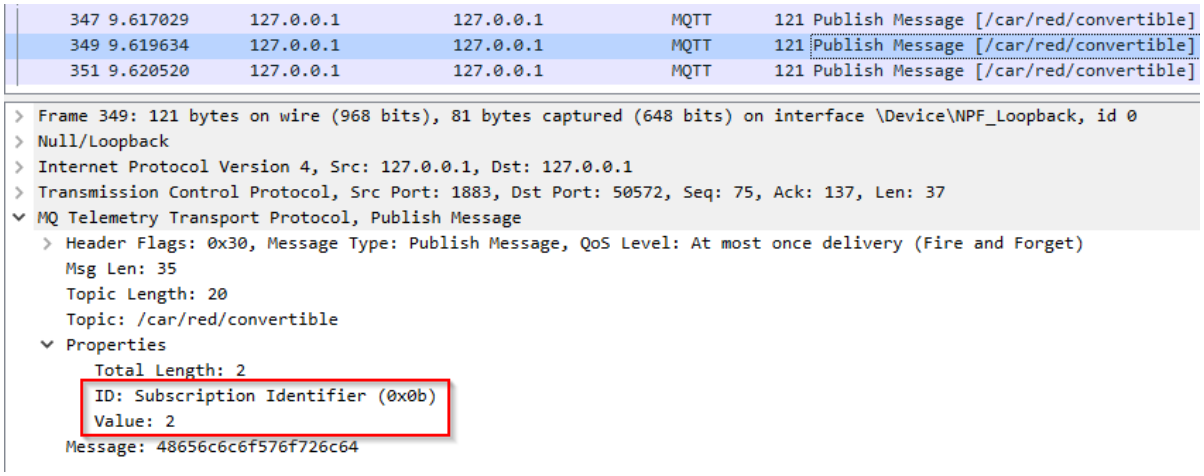
Clients können beim Anlegen einer Subscription einen sogenannten Subscription Identifier angeben. Der Message Broker speichert dann intern ein Mapping zwischen dieser Subscription und dem jeweiligen Identifier. Sobald eine Nachricht vom Broker an den Client gesendet wird, wird der Subscription Identifier mit an den Client übertragen. Der Client hat dann die Möglichkeit, die Nachricht intern anhand des Subscription Identifiers zuzuordnen. Das folgende Schaubild verdeutlicht diesen Zusammenhang nochmals:



In diesem Beispiel erstellt der MQTT-Client auf der linken Seite vier Subscriptions, jeweils auf teilweise „überlappende“ Topics mit Hilfe von Wildcards [▶ 29]. Nun published der rechte MQTT Client eine Nachricht auf dem Topic /car/red/convertible.

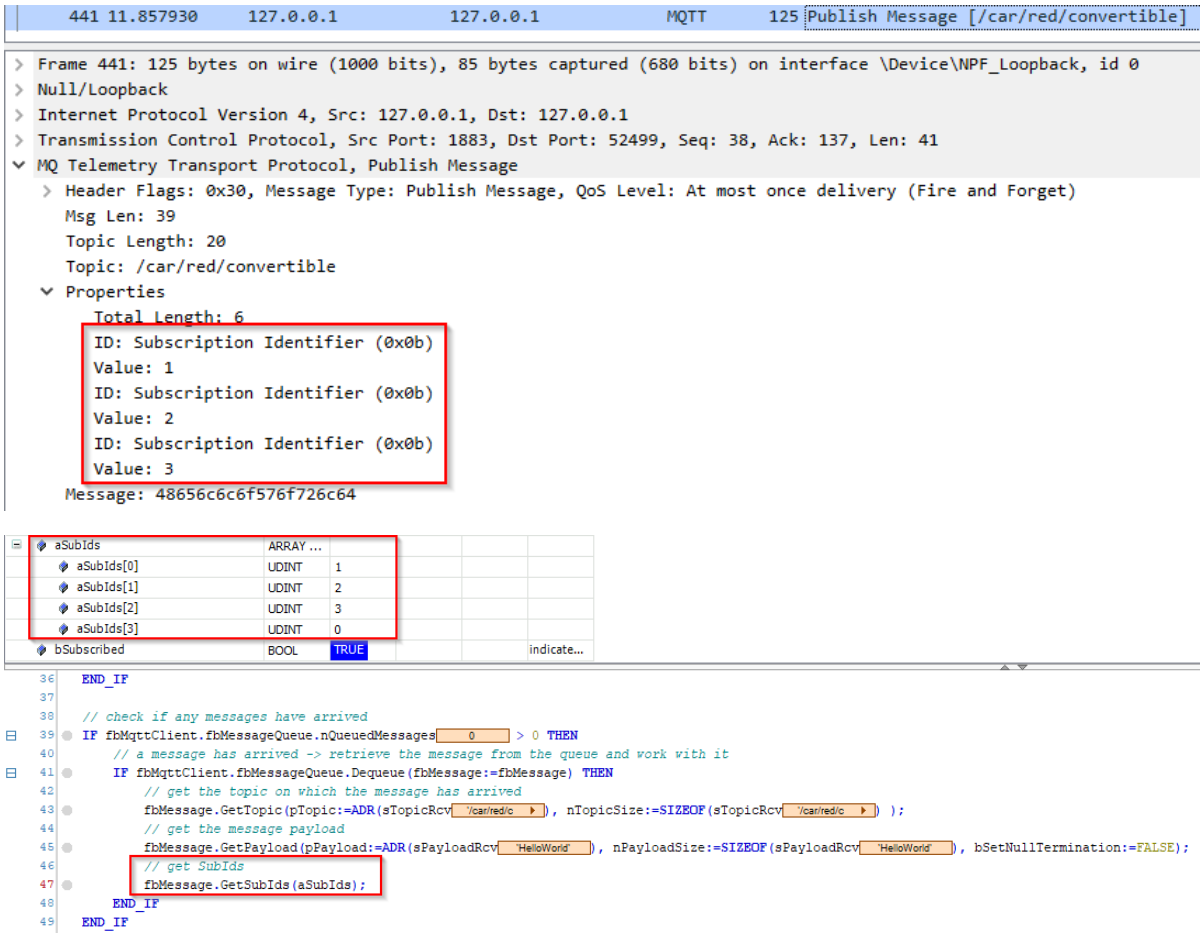
Nun gibt es zwei Möglichkeiten:

1. Unoptimierte Variante: Der Subscriber erhält drei MQTT Nachrichten (Publishes) vom Broker. Jede Nachricht enthält den jeweiligen Subscription Identifier. In der TwinCAT SPS würden auf Basis unseres Samples IotMqttv5Sample [▶ 333] somit drei Nachrichten in der Message Queue landen. Beispiel-Screenshot aus Wireshark beim Empfang der drei Nachrichten:



   Der Mosquitto Broker in der Version 2.0.15 verwendet standardmäßig diese Form der Übertragung.
2. Der Subscriber enthält nur eine MQTT Nachricht vom Broker, wobei diese Nachricht eine Liste mit Subscription Identifier enthält. In obigem Beispiel also die Identifier 1, 2 und 3. In der TwinCAT SPS würde in diesem Fall nur eine Nachricht in der Message Queue landen. Diese Nachricht enthält dann

ein Array der Subscription Identifier. Beispiel-Screenshot aus Wireshark beim Empfang der Nachricht:



Der HiveMQ CE Broker in der Version 2023.3 verwendet standardmäßig diese Form der Übertragung.

Beide Übertragungsarten sind laut MQTT-Spezifikation zulässig.

4.5.9.5 Ablaufintervalle

Session Expiry

Bei MQTT3 kann durch das beim Verbindungsaufbau übergebene Clean Session-Flag festgelegt werden, ob der Message Broker Session-Informationen des Clients nach Abbau der Verbindungen speichert. MQTT5 ersetzt diese Einstellung durch ein Clean Start-Flag und die Einführung eines Session Expiry-Intervalls. Das Clean Start-Flag hat die gleiche Funktion wie Clean Session bei MQTT3. Wenn jedoch die Session des Clients nicht direkt gelöscht werden soll, kann die Ablaufzeit der Speicherung über Session Expiry festgelegt werden (Referenz: ST_IotMqtt5Connect [▶ 97]).

Message Expiry

Wenn eine Nachricht mit QoS 1 oder 2 gepublisht wird und zusätzlich von einem Client mit QoS 1 oder 2 subscribed ist und der Client ohne das Clean Start-Flag verbunden war, wird bei MQTT3 eine Nachricht so lange im Message Broker vorgehalten, bis der Client erneut verbunden ist. Bei MQTT5 ist es möglich, für eine Nachricht eine Ablaufzeit zu definieren. Nach dieser Ablaufzeit wird die Nachricht nicht mehr an zum Zeitpunkt des Versendens nicht verbundene Clients zugestellt.

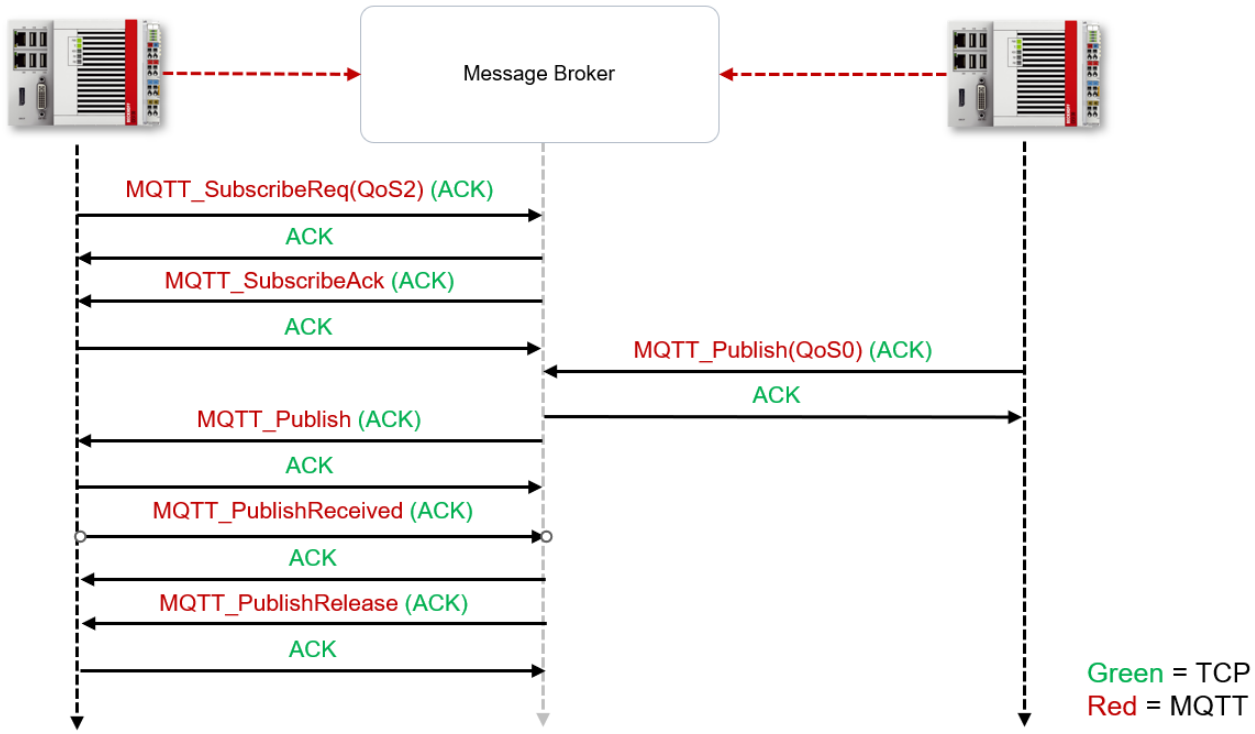
4.5.9.6 Reason codes

Bei MQTT3 gibt es lediglich 10 Return Codes, die bei den Nachrichtentypen CONNACK und SUBACK zurückgegeben wurden. Mit MQTT5 hingegen ist es möglich, in vielen Paketen sogenannte Reason Codes zu übergeben. Ein Reason Code gibt an, dass ein vordefinierter Protokollfehler stattgefunden hat und wird üblicherweise bei Acknowledge-Paketen übermittelt damit Client und Message Broker auf den Fehlerzustand reagieren können.

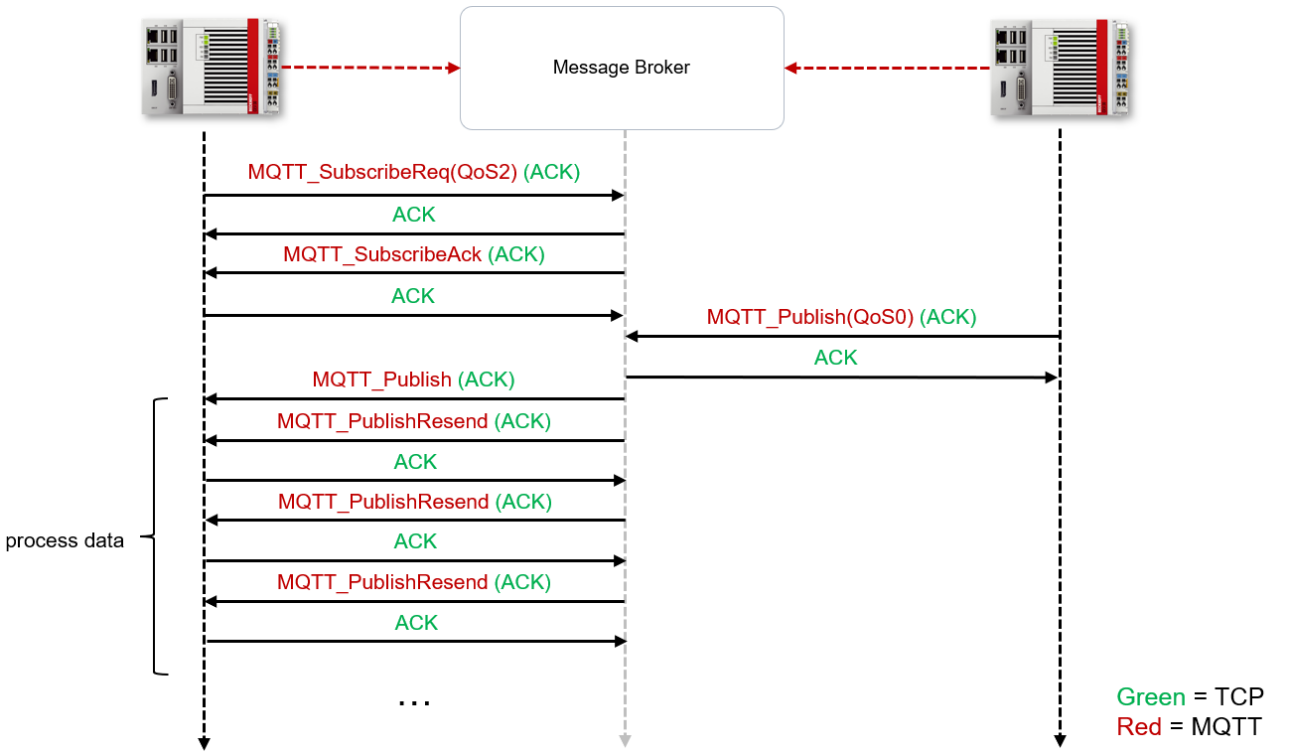
4.5.9.7 Retry-handling QoS

MQTT verwendet TCP als unterlagertes Transportprotokoll. TCP hat die Eigenschaft, dass Nachrichten im Falle einer funktionierenden Verbindung genau einmal und in korrekter Reihenfolge eintreffen. Alle Pakete einer MQTT Verbindung kommen also an der Gegenstelle an. Falls die TCP Verbindung unterbrochen wird, stehen mit QoS [▶ 31] 1 und 2 Optionen zur Verfügung, dass die Nachrichten über mehrere TCP-Verbindungen hinweg zugestellt werden können.

MQTT3 erlaubte bereits eine wiederholte Nachrichtenzustellung bei unterbrechungsfreier TCP Verbindung. Dies konnte jedoch zur Folge haben, dass ein überlasteter MQTT Client im Zweifelsfall noch weiter überlastet wurde, z.B. wenn der Client für die Verarbeitung einer empfangenen Nachricht mehrere Sekunden braucht und in der Zwischenzeit der Message Broker die Nachricht nochmal verschickt weil er das Acknowledgement (noch) nicht erhalten hat.Die folgende Grafik veranschaulicht noch einmal den Empfangsvorgang einer Nachricht aus Sicht des Subscribers, welcher sich mit QoS 2 auf ein Topic am Message Broker subscribed hat:



Wenn der Empfängers nun durch eine Überlastung nicht sofort ein MQTT_PublishReceived Kommando nach dem MQTT_Publish versendet, wird der Message Broker bei MQTT3 einen erneuten Versand des Publishes vorbereiten und auf den Weg schicken. Die TCP-Verbindung ist in diesem Fall weiterhin aktiv. Dies wird in der folgenden Grafik veranschaulicht.



Durch eine neue Erweiterung in MQTT5, wird die wiederholte MQTT Nachrichtenübermittlung (Resend) bei bestehender TCP-Verbindung für Clients und Message Broker unterbunden.

4.5.9.8 Bi-directional disconnect

In MQTT3 konnte ein Client einen ordnungsgemäßen Disconnect mit dem Message Broker durchführen, d.h. der Client hat den Broker über den Verbindungsabbruch informiert. Das Protokoll sah es jedoch nicht vor, dass der Message Broker seinerseits die Clients darüber informieren konnte dass er die Verbindung trennt.

In MQTT5 gibt es nun einen solchen Mechanismus. Der Message Broker kann ein MQTT Disconnect Paket an die Clients verschicken und hierbei auch einen Reason Code für die Trennung der Verbindung angeben.

4.6 Sicherheit

Bei der Betrachtung der Absicherung einer Datenkommunikation lassen sich zwei Ebenen voneinander unterscheiden. Zum einen die Absicherung des Transportkanals [▶ 42] und zum anderen die Absicherung auf Applikationsebene [▶ 49].

4.6.1 Transportebene

Für die sichere Übertragung von Daten wird im TwinCAT IoT-Treiber der weltweit gängige Standard Transport Layer Security (TLS) verwendet. Das folgende Kapitel beschreibt den TLS-Kommunikationsablauf exemplarisch am Beispiel von TLS-Version 1.2.

TLS ist ein Standard, welcher eine Kombination aus symmetrischer und asymmetrischer Kryptographie darstellt und versendete Daten vor unbefugtem Zugriff und Manipulation durch Dritte schützt. Außerdem wird die Authentifizierung der Kommunikationsteilnehmer zur gegenseitigen Identitätsprüfung von TLS unterstützt.

Inhalt dieses Kapitels

Die folgenden Informationen in diesem Kapitel beziehen sich in allgemeiner Weise auf den TLS-Kommunikationsablauf und haben keinen Bezug zu der Implementierung in TwinCAT. Sie sollen lediglich für ein grundlegendes Verständnis sorgen, um den in den folgenden Unterkapiteln erläuterten Bezug zu der TwinCAT-Implementierung besser nachvollziehen zu können.

Unterstützte Funktionen

Der TwinCAT IoT -Treiber ermöglicht die Verwendung der folgenden TLS-Funktionen.

Funktion	Beschreibung
Selbst signierte Clientzertifikate	Verwendung eines selbst-signierten Clientzertifikats zur Authentifizierung am Message Broker.
CA-signierte Clientzertifikate	Verwendung eines CA-signierten Clientzertifikats zur Authentifizierung am Message Broker. Das CA-Zertifikat kann zum Herstellen einer Vertrauensstellung ebenfalls angegeben werden.
Zertifikatssperrlisten	Verwendung von Zertifikatssperrlisten (englisch: Certificate Revocation Lists (CRL)).
Pre-Shared Key (PSK)	Verwendung eines Pre-Shared Key (PSK) zur Authentifizierung am Message Broker.

Definition Cipher-Suite

Eine Cipher Suite ist per Definition eine Zusammensetzung von Algorithmen (Schlüsselaustausch, Authentifizierung, Verschlüsselung, MAC) zur Verschlüsselung. Auf diese einigen sich Client und Server während des TLS-Verbindungsaufbaus. Weitere Informationen zu Cipher Suites entnehmen Sie bitte der Fachliteratur.

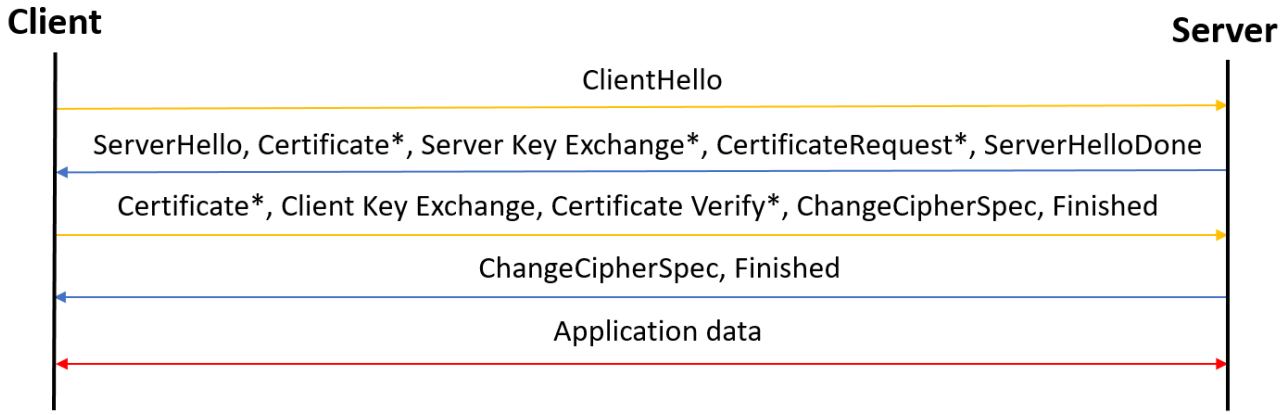
TLS-Kommunikationsablauf

Am Anfang einer Kommunikation mit TLS-Verschlüsselung steht der sogenannte TLS-Handshake zwischen Server und Client. Während des Handshakes wird asymmetrische Kryptographie verwendet, nach erfolgreichem Abschluss des Handshakes kommunizieren Server und Client mit symmetrischer Kryptographie, da diese um ein Vielfaches schneller ist als asymmetrische Kryptographie.

Es gibt drei verschiedene Arten der Authentisierung für das TLS-Protokoll:

- Der Server weist sich per Zertifikat aus (siehe Server-Zertifikat [▶ 45])
- Client und Server weisen sich per Zertifikat aus (siehe Client/Server-Zertifikat [▶ 46])
- Pre-Shared-Keys (siehe Pre-Shared-Keys [▶ 47])

Über Vor- und Nachteile der verschiedenen Authentisierungsarten informieren Sie sich bitte in der Fachliteratur.



Beispielhafte Erläuterung anhand von RSA

Alle mit * gekennzeichneten Nachrichten sind optional und werden nicht zwingend benötigt. Die nachfolgenden Schritte beziehen sich auf das RSA-Verfahren und haben keine allgemeine Gültigkeit für andere Verfahren.

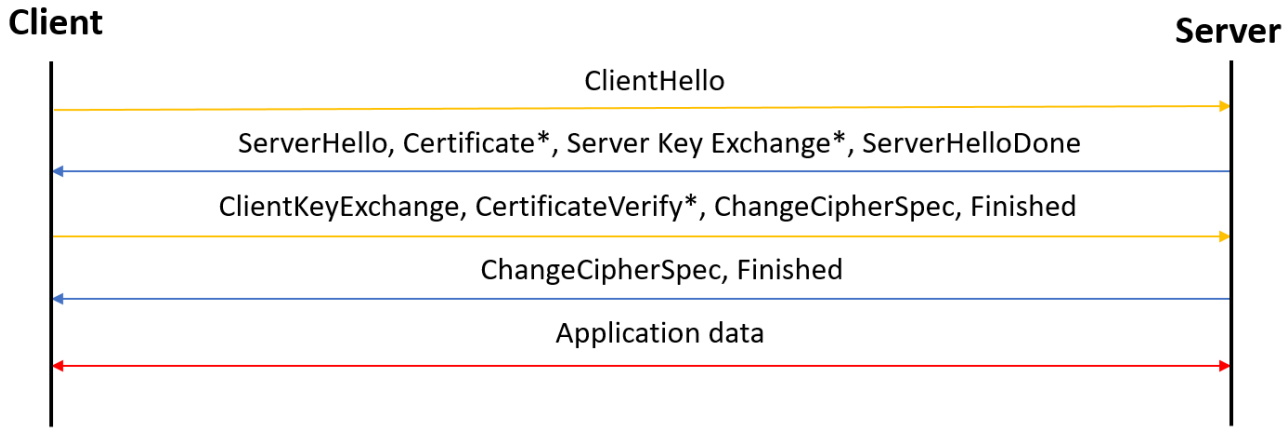
Die folgende Tabelle erläutert die einzelnen Schritte aus dem oben dargestellten Kommunikationsablauf.

Schritt	Beschreibung
ClientHello	Der Client initiiert einen Verbindungsaufbau zum Server. Dabei werden unter anderem die verwendete TLS-Version, eine Zufallsfolge von Bytes (Client Random) und die vom Client unterstützten Cipher Suites übertragen.
ServerHello	Der Server sucht sich aus den vom Client offerierten Cipher Suites eine aus und legt diese für die Kommunikation fest. Besteht keine Schnittmenge zwischen den vom Client und Server unterstützten Cipher Suites, wird der TLS-Verbindungsaufbau abgebrochen. Zusätzlich kommuniziert auch der Server eine Zufallsfolge von Bytes (Server Random).
Certificate	Der Server präsentiert dem Client sein Zertifikat, damit der Client verifizieren kann, dass der Server auch der erwartete Server ist. Vertraut der Client dem Server-Zertifikat nicht, wird der TLS-Verbindungsaufbau abgebrochen. Das Server-Zertifikat enthält zusätzlich den öffentlichen Schlüssel des Servers.
ServerKeyExchange	Bei bestimmten Schlüsselaustauschalgorithmen reichen die Informationen aus dem Zertifikat nicht aus, damit der Client das sogenannte Pre-Master Secret erzeugen kann. In diesem Fall werden die fehlenden Informationen mittels Server Key Exchange übertragen.
CertificateRequest	Der Server fordert vom Client ein Zertifikat an, um die Identität des Clients verifizieren zu können.
ServerHelloDone	Der Server teilt dem Client mit, dass sein Versenden der initialen Informationen beendet ist.
Certificate	Der Client kommuniziert sein Zertifikat inklusive des öffentlichen Schlüssels an den Server. Der Ablauf ist der gleiche wie in entgegengesetzter Richtung: Vertraut der Server dem vom Client versendeten Zertifikat nicht, wird der Verbindungsaufbau abgebrochen.
ClientKeyExchange	Der Client verwendet den öffentlichen Schlüssel des Servers, um durch asymmetrische Verschlüsselung ein von ihm generiertes Pre-Master Secret verschlüsselt an den Server zu schicken. Aus diesem Pre-Master Secret, dem Server Random und dem Client Random wird dann der symmetrische Schlüssel berechnet, der nach dem Verbindungsaufbau für die Kommunikation verwendet wird.
CertificateVerify	Der Client signiert die bisherigen Nachrichten des Handshakes mit seinem privaten Schlüssel. Da der Server den öffentlichen Schlüssel des Clients durch das Versenden des Zertifikats erhalten hat, kann er verifizieren, dass das präsentierte Zertifikat auch wirklich dem Client „gehört“.
ChangeCipherSpec	Der Client teilt dem Server mit, dass er auf symmetrische Kryptographie umsteigt. Jede Nachricht vom Client an den Server ab hier ist signiert und verschlüsselt.
Finished	Der Client teilt dem Server verschlüsselt mit, dass der TLS-Verbindungsaufbau auf seiner Seite beendet ist. Die Nachricht enthält einen Hash und einen MAC über die bisherigen Nachrichten des Handshakes.

Schritt	Beschreibung
ChangeCipherSpec	Der Server entschlüsselt das Pre-Master Secret, das der Client mit seinem öffentlichen Schlüssel verschlüsselt hat. Da der Server der Einzige ist, der seinen privaten Schlüssel besitzt, kann nur der Server dieses Pre-Master Secret entschlüsseln. So wird sichergestellt, dass der symmetrische Schlüssel nur Client und Server bekannt ist. Anschließend berechnet auch der Server den symmetrischen Schlüssel aus Pre-Master Secret und den beiden Zufallsfolgen und teilt dem Client mit, dass auch er jetzt mit symmetrischer Kryptographie kommuniziert. Jede Nachricht vom Server an den Client ab hier ist signiert und verschlüsselt. Durch die Generierung des symmetrischen Schlüssels kann der Server die Finished-Nachricht des Clients entschlüsseln und sowohl Hash als auch MAC verifizieren. Sollte diese Verifizierung fehlschlagen, wird die Verbindung abgebrochen.
Finished	Der Server teilt dem Client mit, dass der TLS-Verbindungsaufbau auf seiner Seite ebenfalls beendet ist. Die Nachricht enthält so wie beim Client einen Hash und einen MAC über die bisherigen Nachrichten des Handshakes. Auf der Seite des Clients wird dann dieselbe Verifikation vollzogen wie beim Server, auch hier gilt: Wenn Hash und MAC nicht erfolgreich entschlüsselt werden, wird die Verbindung abgebrochen.
ApplicationData	Client und Server kommunizieren nach Abschluss des TLS-Verbindungsaufbaus mit symmetrischer Kryptographie.

4.6.1.1 Server-Zertifikat

Dieser Abschnitt behandelt den Fall, dass nur der Client das Server-Zertifikat verifizieren will, der Server aber das Client-Zertifikat nicht. Der Kommunikationsablauf aus dem Kapitel Transportebene [▶ 42] verkürzt sich zu folgendem Ablauf.



Verifikation des Server-Zertifikats

Das Server-Zertifikat wird auf Client-Seite verifiziert. Dabei wird überprüft, ob es von einer bestimmten Certificate Authority signiert ist. Ist das nicht der Fall, wird die Verbindung vom Client abgebrochen, da er dem Server dann nicht vertraut.

Verwendung in TwinCAT

In TwinCAT wird in dem Fall der Dateipfad zum CA-Zertifikat angegeben (.PEM oder .DER-Datei) oder der Inhalt der .PEM-Datei als String. Im IoT-Treiber wird dann das vom Server präsentierte Zertifikat überprüft. Wenn die Zertifikatskette nicht von der angegebenen CA signiert ist, wird die Verbindung zum Server abgebrochen. Der nachfolgende Code stellt nur exemplarisch die beschriebenen Verbindungsparameter dar. Der Beispielcode bezieht sich auf den HTTP-Client, den MQTT-Client und den WebSocket-Client, exemplarisch wird der HTTP-Client verwendet.

```
PROGRAM MAIN
VAR
    fbClient : FB_IotHttpClient;
END_VAR
```

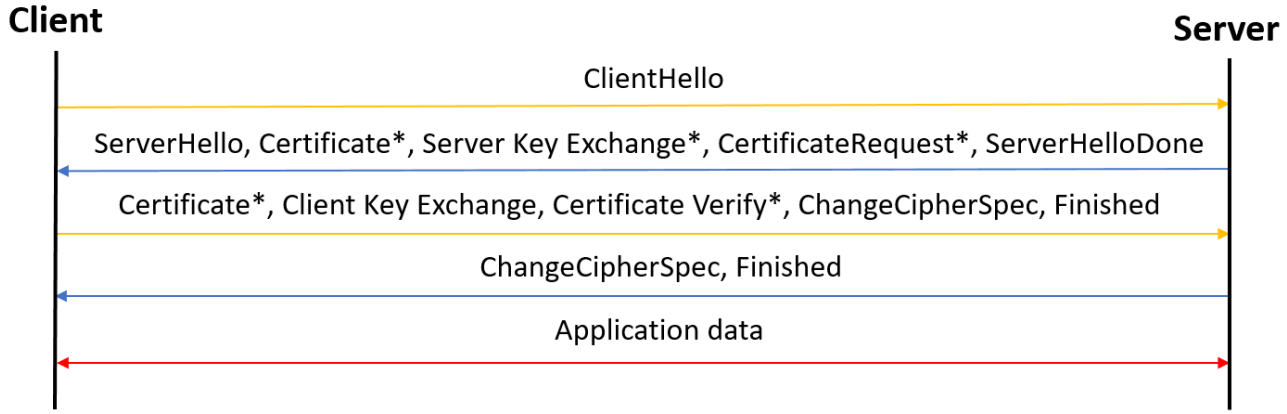
```
fbClient.stTLS.sCA:= 'C:\TwinCAT\3.1\Config\Certificates\someCA.pem';
```

Hat der Benutzer das CA-Zertifikat nicht zur Verfügung, kann trotzdem eine Verbindung hergestellt werden. Dazu gibt es eine boolsche Variable, mit der TwinCAT die Verifikation des Server-Zertifikats verboten wird. Die Verbindung wird zwar mit dem öffentlichen Schlüssel des unverifizierten Server-Zertifikats verschlüsselt hergestellt, ist aber anfälliger gegen Man-in-the-middle-Attacken.

```
fbClient.stTLS.sCA.bNoServerCertCheck:= TRUE;
```

4.6.1.2 Client/Server-Zertifikat

In diesem Abschnitt wird der Fall betrachtet, dass sowohl Client- als auch Server-Zertifikat verifiziert werden. Der im Vergleich zum Server-Zertifikat [▶ 45]-Kapitel leicht abgeänderte Kommunikationsablauf wird in der folgenden Grafik visualisiert. Die einzelnen Schritte des TLS-Verbindungsaufbaus sind im Kapitel Transportebene [▶ 42] beschrieben.



Verwendung in TwinCAT

In TwinCAT wird im Falle der Verwendung eines Client-Zertifikats ebenso wie beim CA-Zertifikat der Dateipfad (.PEM- oder .DER-Datei) oder der Inhalt der .PEM-Datei als String übergeben. Dieses Zertifikat präsentiert TwinCAT als Client dann dem Server. Für das Certificate Verify muss zusätzlich der private Schlüssel des Clients referenziert werden. Optional kann im Falle eines Passwortschutzes für den privaten Schlüssel auch dieses Passwort übergeben werden. Der Beispielcode bezieht sich auf den HTTP-Client, den MQTT-Client und den WebSocket-Client, exemplarisch wird der HTTP-Client verwendet.

```
PROGRAM MAIN
VAR
    fbClient : FB_IotHttpClient;
END_VAR
```

```
fbClient.stTLS.sCA:= 'C:\TwinCAT\3.1\Config\Certificates\someCA.pem';
fbClient.stTLS.sCert:= 'C:\TwinCAT\3.1\Config\Certificates\someCRT.pem';
fbClient.stTLS.sKeyFile:= 'C:\TwinCAT\3.1\Config\Certificates\someprivatekey.pem.key';
fbClient.stTLS.sKeyPwd:= 'yourkeyfilepasswordhere';
```

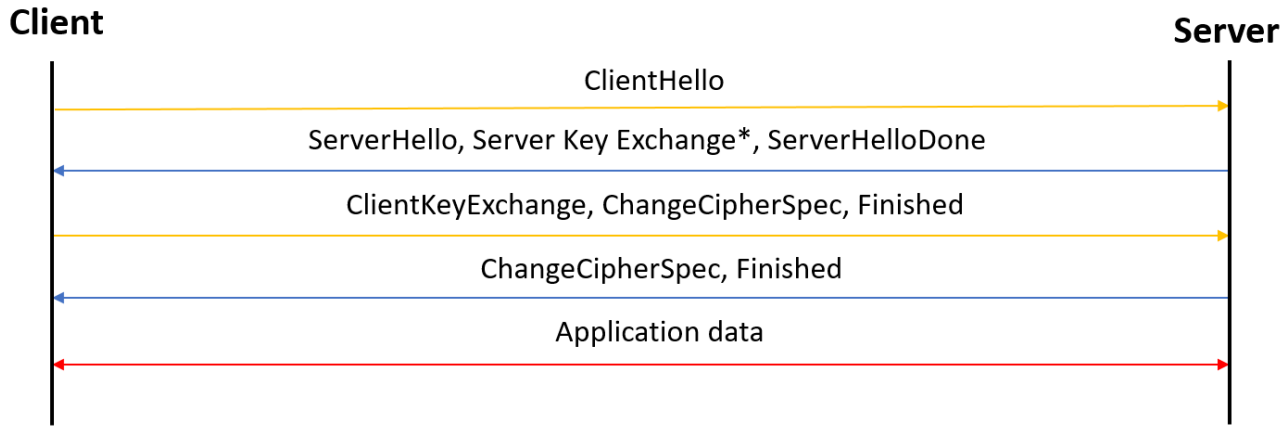
In dem Fall, dass ein Client-Zertifikat gesetzt ist, muss auch ein CA-Zertifikat zur Validierung des Server-Zertifikats gesetzt werden. Das ist durch das Verhalten des im IoT-Treiber verwendeten Security-Frameworks bedingt.

Wenn in diesem Fall die Validierung des Server-Zertifikats abgeschaltet werden soll, kann zusätzlich ein Flag zum Überspringen der Validierung gesetzt werden. Es ist aber nicht möglich, das CA-Zertifikat wegzulassen.

4.6.1.3 Pre-Shared-Keys

Standardmäßig werden beim TLS-Verbindungsaufbau asymmetrische Schlüsselpaare verwendet. Asymmetrische Kryptographie verbraucht höhere Rechenleistung, daher kann es in Umgebungen mit wenig CPU-Leistung eine Möglichkeit sein, Pre-Shared Keys (PSK) zu verwenden. Pre-Shared-Keys sind vorher geteilte, symmetrische Schlüssel.

Verglichen mit dem Kommunikationsablauf bei asymmetrischer Verschlüsselung fällt bei Verwendung von PSK das Zertifikat weg. Client und Server müssen sich über die sogenannte Identität auf einen PSK einigen. Der PSK ist per Definition vorher beiden Parteien bekannt.



Server Key Exchange: In dieser optionalen Nachricht kann der Server dem Client einen Hinweis auf die Identität des verwendeten PSK geben.

Client Key Exchange: Der Client gibt die Identity des PSK an, mit dem die Verschlüsselung durchgeführt werden soll.

Verwendung in TwinCAT

In TwinCAT wird die Identität des PSK als String angegeben, der PSK selbst wird als ByteArray in der Steuerung abgespeichert. Zusätzlich wird noch die Länge des PSK angegeben. Der Beispielcode bezieht sich auf den HTTP-Client, den MQTT-Client und den WebSocket-Client, exemplarisch wird der HTTP-Client verwendet.

```
PROGRAM MAIN
VAR
    fbClient : FB_IotHttpClient;
    cMyPskKey : ARRAY[1..64] OF BYTE := [16#1B, 16#D0, 16#6F, 16#D2, 16#56, 16#16, 16#7D, 16#C1, 16#
E8, 16#C7, 16#48, 16#2A, 16#8E, 16#F5, 16#FF];
END_VAR
```

```
fbClient.stTLS.sPskIdentity:= identityofPSK';
fbClient.stTLS.aPskKey:= cMyPskKey;
fbClient.stTLS.nPskKeyLen:= 15;
```

4.6.1.4 Unterstützte Cipher-Suites

Der TwinCAT IoT-Treiber unterstützt eine sichere Datenübertragung unter Verwendung des TLS-Standards. Nachfolgend finden Sie eine Übersicht zu allen vom IoT-Treiber unterstützten Cipher-Suites, abhängig von der TwinCAT Version.

TwinCAT 3.1 Build 4024.x

Cipher-Suite
AES128-GCM-SHA256
AES128-SHA
AES128-SHA256
AES256-SHA
AES256-SHA256
DES-CBC3-SHA
DHE-RSA-AES128-GCM-SHA256
DHE-RSA-AES128-SHA
DHE-RSA-AES128-SHA256
DHE-RSA-AES256-SHA
DHE-RSA-AES256-SHA256
ECDHE-ECDSA-AES128-GCM-SHA256
ECDHE-ECDSA-AES128-SHA
ECDHE-ECDSA-AES128-SHA256
ECDHE-ECDSA-AES256-SHA
ECDHE-ECDSA-DES-CBC3-SHA
ECDHE-RSA-AES128-GCM-SHA256
ECDHE-RSA-AES128-SHA
ECDHE-RSA-AES128-SHA256
ECDHE-RSA-AES256-SHA
ECDHE-RSA-DES-CBC3-SHA
EDH-RSA-DES-CBC3-SHA
PSK-3DES-EDE-CBC-SHA
PSK-AES128-CBC-SHA
PSK-AES128-CBC-SHA256
PSK-AES128-GCM-SHA256
PSK-AES256-CBC-SHA

TwinCAT 3.1 Build 4026.x

Cipher-Suite
AES128-GCM-SHA256
AES128-SHA
AES128-SHA256
AES256-GCM-SHA384
AES256-SHA
AES256-SHA256
DHE-RSA-AES128-GCM-SHA256
DHE-RSA-AES128-SHA
DHE-RSA-AES128-SHA256
DHE-RSA-AES256-GCM-SHA384
DHE-RSA-AES256-SHA
DHE-RSA-AES256-SHA256
ECDHE-ECDSA-AES128-GCM-SHA256
ECDHE-ECDSA-AES128-SHA
ECDHE-ECDSA-AES128-SHA256
ECDHE-ECDSA-AES256-GCM-SHA384
ECDHE-ECDSA-AES256-SHA
ECDHE-ECDSA-AES256-SHA384
ECDHE-RSA-AES128-GCM-SHA256
ECDHE-RSA-AES128-SHA
ECDHE-RSA-AES128-SHA256
ECDHE-RSA-AES256-GCM-SHA384
ECDHE-RSA-AES256-SHA
ECDHE-RSA-AES256-SHA384
PSK-AES128-CBC-SHA
PSK-AES128-CBC-SHA256
PSK-AES128-GCM-SHA256
PSK-AES256-CBC-SHA
PSK-AES256-CBC-SHA384
PSK-AES256-GCM-SHA384

4.6.2 Applikationsebene

Benutzeridentität

Die MQTT-Spezifikation definiert auf Applikationsebene lediglich eine Benutzername/Passwort-Authentifizierung, welche beim Verbindungsaufbau erfolgen kann. Weitere Mechanismen sind nicht spezifiziert und können applikativ gehandhabt werden. Manche Message Broker ermöglichen zum Beispiel auch die Verwendung des CommonName Attributs aus dem Clientzertifikat als Benutzeridentität, wodurch dann zum Beispiel Zugriffsrechte auf Topic-Ebene definiert werden können.

JSON Web Token (JWT)

JSON Web Token (JWT) sind ein offener Standard (nach RFC 7519), welche ein kompaktes und sich selbst beschreibendes Format definieren, um Informationen sicher zwischen Kommunikationsteilnehmern in Form eines JSON Objekts zu übertragen. Die Authentizität der übertragenen Information kann hierbei verifiziert und sichergestellt werden, da ein JWT mit einer digitalen Signatur versehen wird. Die Signatur kann hierbei über ein Shared Secret (via HMAC Algorithmus) oder einen Public/Private Key (via RSA) erfolgen.

Das am weitesten verbreitete Anwendungsbeispiel für JWT ist die Autorisierung eines Geräts oder Benutzers an einem Service. Sobald sich ein Benutzer an dem Service angemeldet hat, beinhalten alle weiteren Anfragen an den Service das JWT. Anhand des JWT kann der Service dann entscheiden, auf welche weiteren Dienste oder Ressourcen der Benutzer zugreifen darf. Hierdurch können zum Beispiel Single Sign On Lösungen in Cloud-Diensten realisiert werden.

Die SPS-Bibliothek Tc3_JsonXml stellt über die Methode FB_JwtEncode die Möglichkeit bereit, ein JSON Web Token zu erzeugen und zu signieren. Das Token kann dann z.B. im Payload der Nachricht oder auch in einem User Property [▶ 35] verschickt werden, um bei der empfangenden Applikation eine Validierung der Nachricht zu ermöglichen.

4.7 Neuparametrierung

In bestimmten Fällen kann eine Neuparametrierung des MQTT-Clients während des laufenden Betriebes durch einen Online-Change notwendig sein. Dabei kann diese Neuparametrierung entweder automatisch im Programmablauf erfolgen oder bei Bedarf manuell getriggert werden. Dies ist aber von der Implementierung abhängig.

Beispielhafte Anwendungsfälle für eine Neuparametrierung können der Austausch eines in Kürze ablaufenden Tokens, zu erneuernde Zertifikate oder eine geänderte IP-Adresse des MQTT-Brokers sein. Im Folgenden wird der notwendige Ablauf für eine Neuparametrierung eines verbundenen MQTT-Clients beschrieben.

Damit der TwinCAT-MQTT-Client mit einem Message Broker verbunden werden kann, muss im Hintergrund zyklisch die Execute [▶ 60]-Methode aufgerufen werden (siehe FB_IotMqttClient [▶ 57]). Bei einem Programmstart wird durch den Aufruf dieser Methode zunächst die Parametrierung der MQTT-Client-Instanz vorgenommen, nach deren Abschluss eine Verbindung zum Broker hergestellt wird. Um eine Neuparametrierung vorzunehmen, müssen der Aufruf der Execute [▶ 60]-Methode negiert erfolgen, anschließend die Parameter neu gesetzt werden und der Aufruf dann wieder wie gewohnt stattfinden. Das folgende Code-Snippet zeigt eine einfache Neuparametrierung anhand eines Triggers.

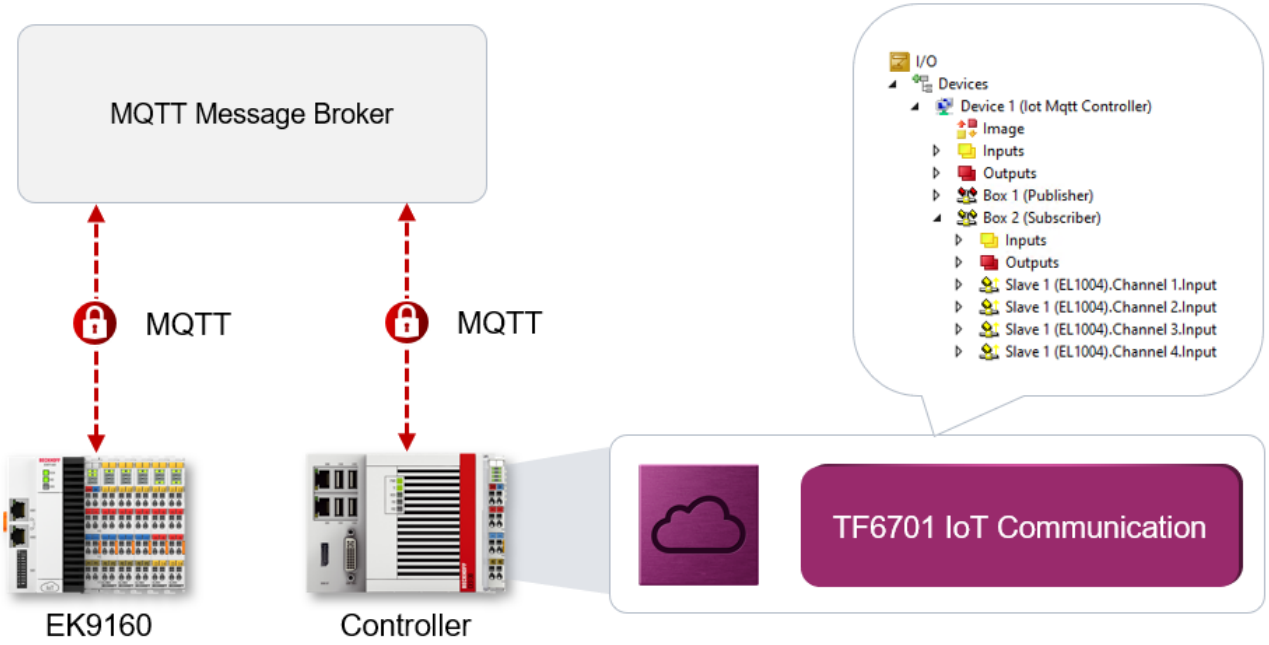
```
fbMqttClient.Execute(bConnect);

IF bReset THEN
    bConnect:=FALSE;
    fbMqttClient.ClientId:= 'MyTcMqttClient35';
    fbMqttClient.sHostName:= '192.168.35.35';
    bReset:=FALSE;
ELSE
    bConnect:=TRUE;
END_IF
```

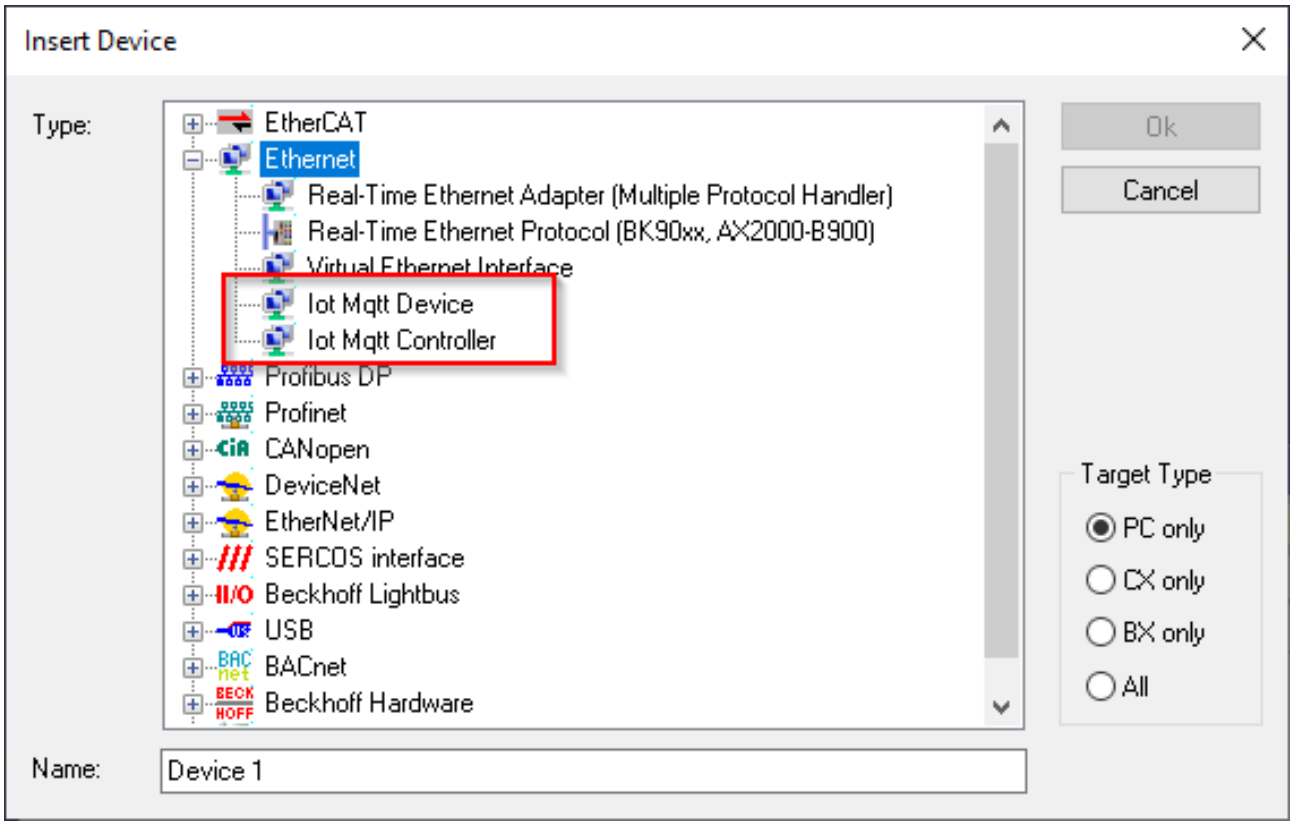
Durch die Trigger-Variable bReset wird der Aufruf der Execute-Methode negiert, sodass im Anschluss die Parameter für die Client-ID und den Hostnamen neu gesetzt werden können. Ist die Trigger-Variable wieder auf FALSE gesetzt, wird die Execute-Methode wieder zyklisch aufgerufen.

4.8 I/O Gerät

Zusätzlich zur SPS-Bibliothek Tc3_IotBase stehen mit dem IoT MQTT Controller und IoT MQTT Device zwei TwinCAT I/O Geräte zur Verfügung, mit deren Hilfe sich eine MQTT-basierte Kommunikationsverbindung zwischen zwei TwinCAT Systemen aufbauen lässt. Alternativ lässt sich auch ein EK9160 IoT-Koppler in diese Art von Kommunikation integrieren.



Die beiden Gerätetypen können über den entsprechenden Dialog im TwinCAT I/O Bereich hinzugefügt werden und befinden sich dort innerhalb der Kategorie Ethernet.



Bei einem IoT MQTT Device werden dessen Symbolinformationen zu allen konfigurierten Variablen des Prozessabbilds auf dem Message Broker in einem bestimmten Topic als Retain [▶ 33]-Nachricht abgelegt.

Ein IoT MQTT Controller hat dann die Möglichkeit, diese Symbolinformationen einzuscannen und passende Variablen in seinem eigenen Prozessabbild anzulegen. Der EK9160 ist hierbei automatisch immer ein IoT MQTT Device.

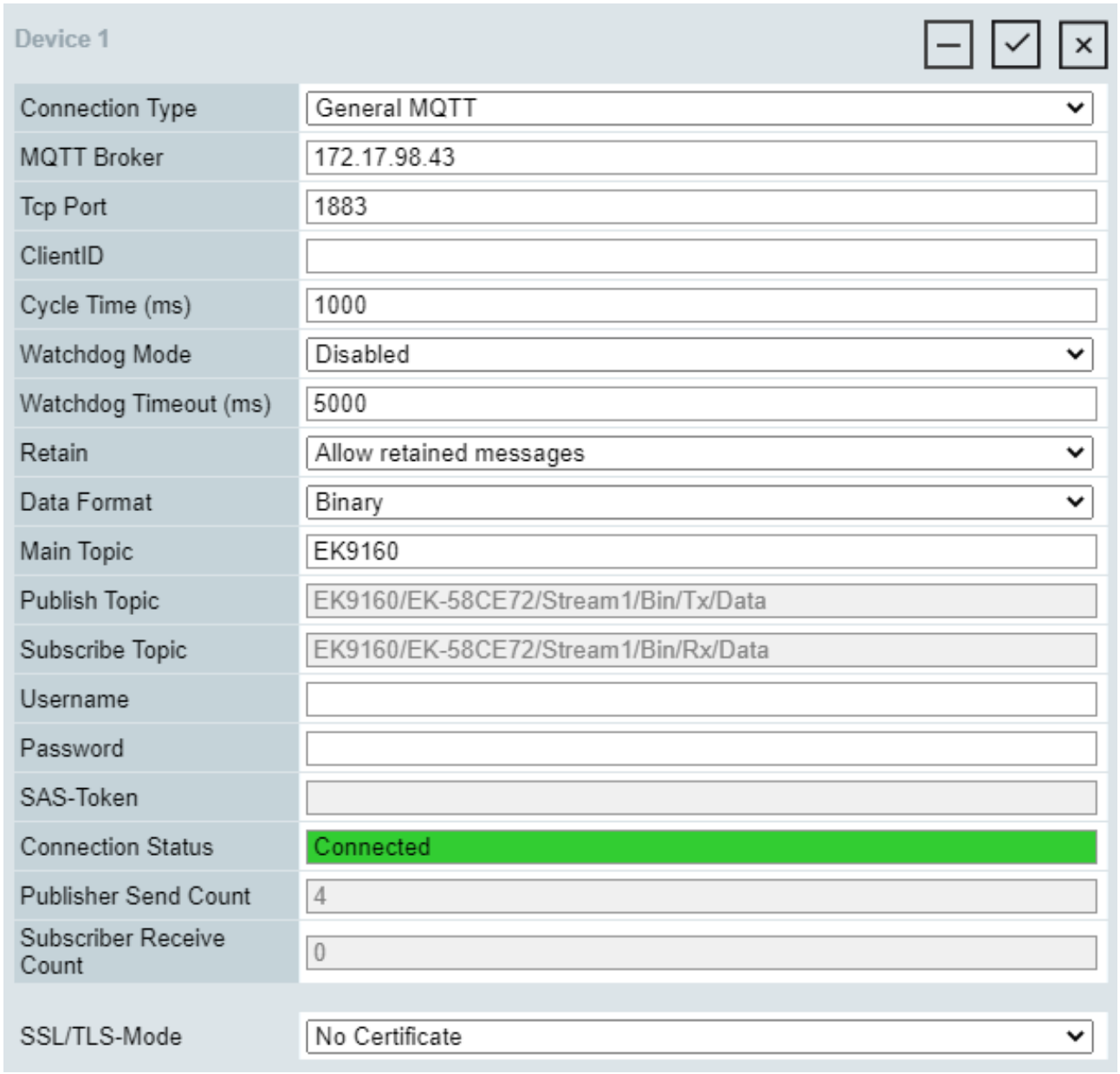
Datenformat

Das Datenformat für diese Art der MQTT Kommunikation muss auf „Binär“ eingestellt werden.

Konfiguration des EK9160

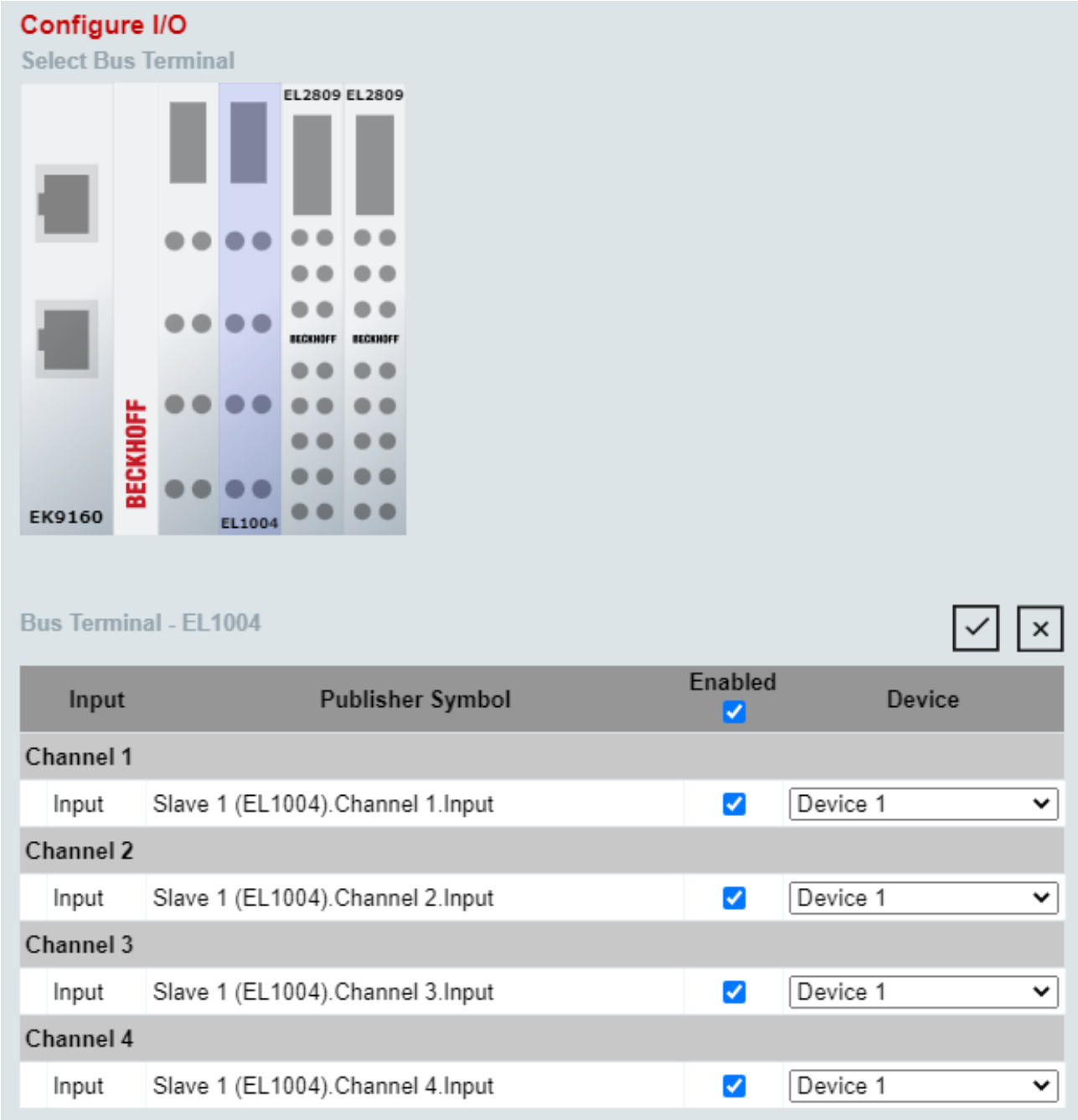
Die Konfiguration des EK9160 als IoT MQTT Device erfolgt automatisch im Hintergrund, sobald das Gerät für die Verbindung mit einem Message Broker konfiguriert wird. Als Voraussetzung muss als Datenformat „Binary“ gewählt, sowie Retain-Nachrichten aktiviert werden.

Der folgende Screenshot zeigt den entsprechenden Ausschnitt aus der EK9160 Konfigurationswebseite.



Alle I/O Klemmen sollen in diesem Beispiel für die Kommunikationsverbindung mit dem Message Broker aktiviert werden.

Der folgende Screenshot zeigt diesen Vorgang exemplarisch auf der Konfigurationsoberfläche.

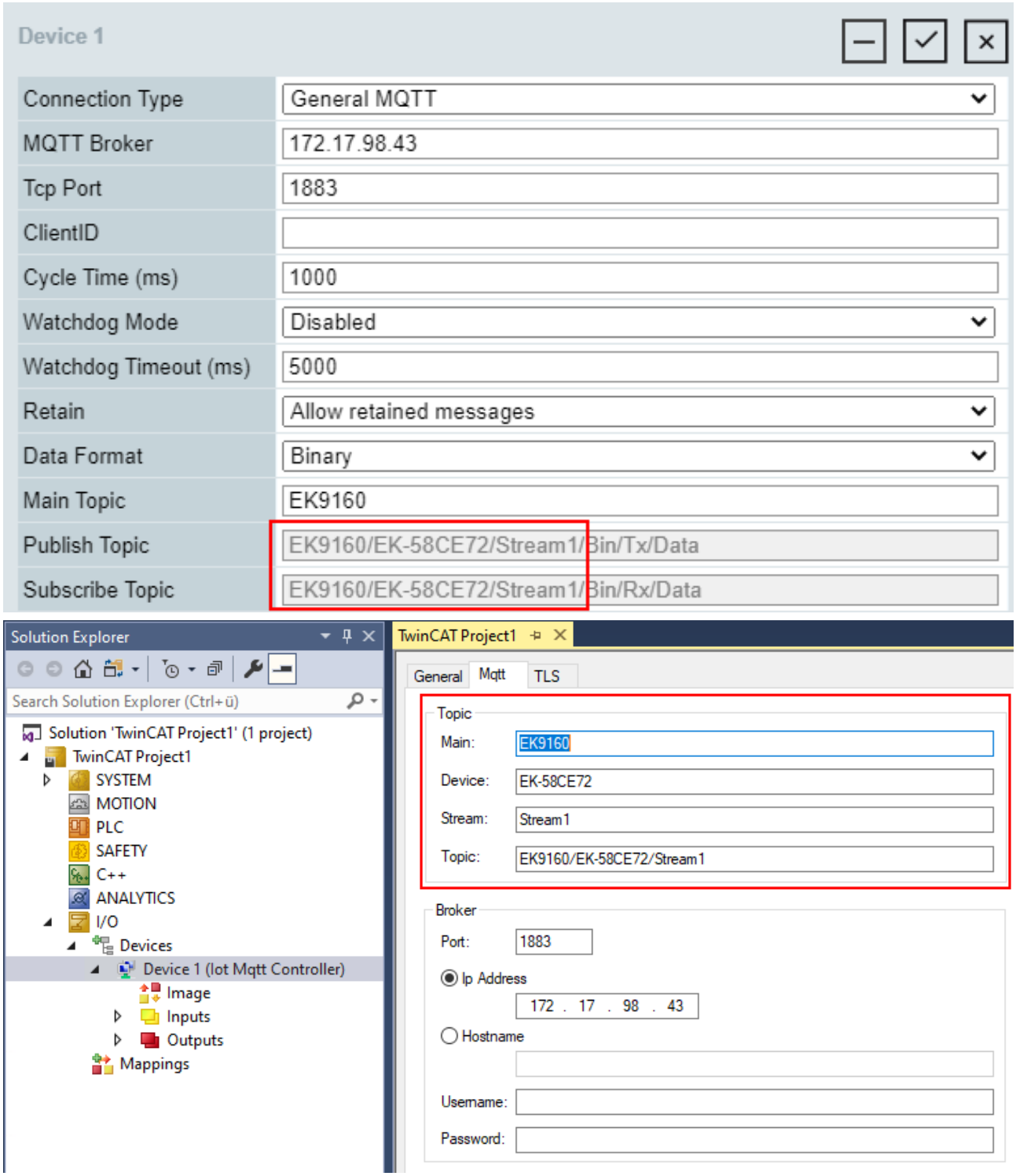


TwinCAT als IoT MQTT Device

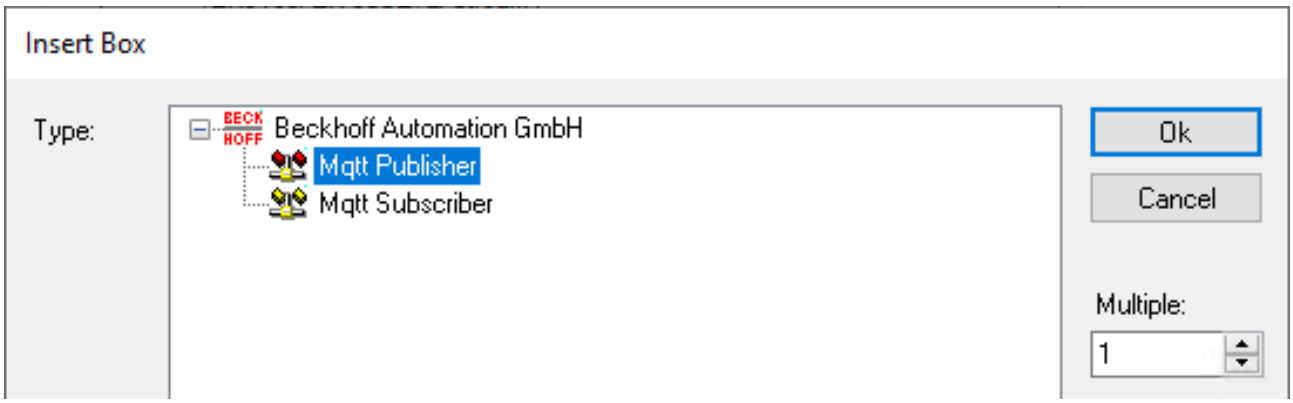
Neben dem EK9160 kann auch ein TwinCAT-System als IoT MQTT Device konfiguriert werden. Die entsprechenden Konfigurationsschritte müssen in diesem Fall über den TwinCAT I/O Bereich manuell durchgeführt werden. Das IoT MQTT Device verhält sich dann vom weiteren Ablauf analog zum EK9160.

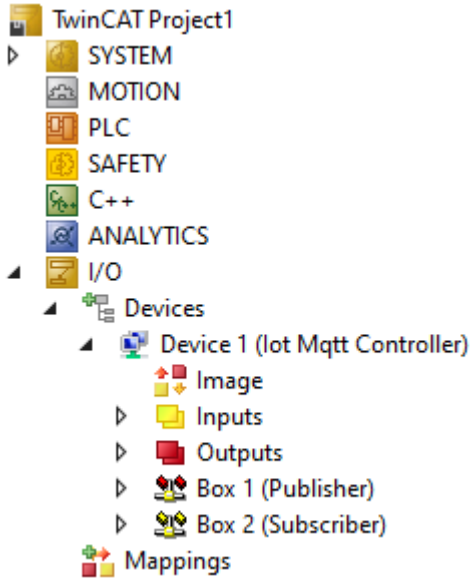
Konfiguration von TwinCAT

Damit TwinCAT die Symbolinformationen und Prozesswerte vom EK9160 verarbeiten kann, muss im I/O Bereich von TwinCAT ein IoT MQTT Controller angelegt und für die Verbindung mit dem Message Broker konfiguriert werden. Entscheidend ist hierbei, dass die Felder **Main Topic**, **Device** und **Stream** mit der Konfiguration des EK9160 übereinstimmen. Der folgende Screenshot veranschaulicht diesen Vorgang.

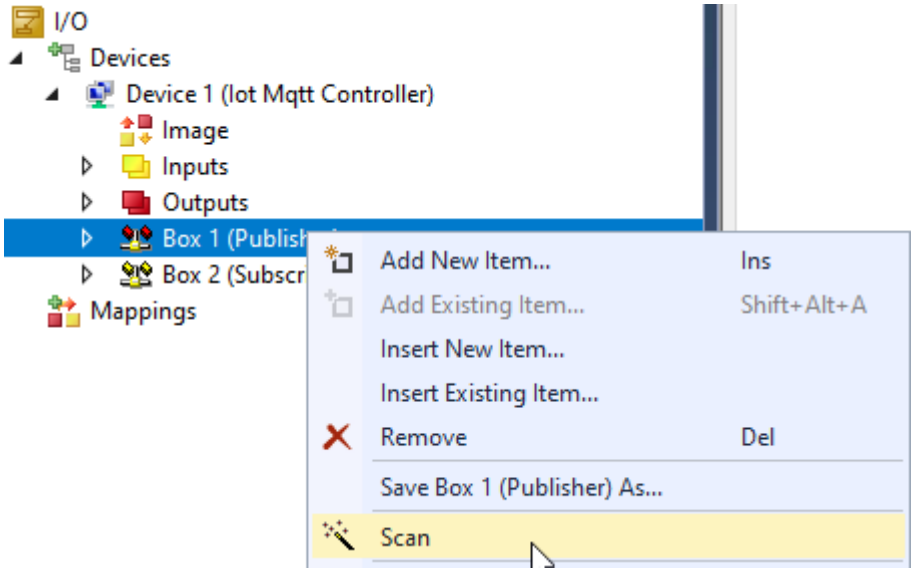


Unterhalb des IoT MQTT Controllers können anschließend Publisher und Subscriber angelegt werden, je nachdem, ob man die Aus- oder Eingangsklemmen einscannen möchte. Eingangsklemmen werden hierbei über den Publisher bedient und Ausgangsklemmen über den Subscriber.

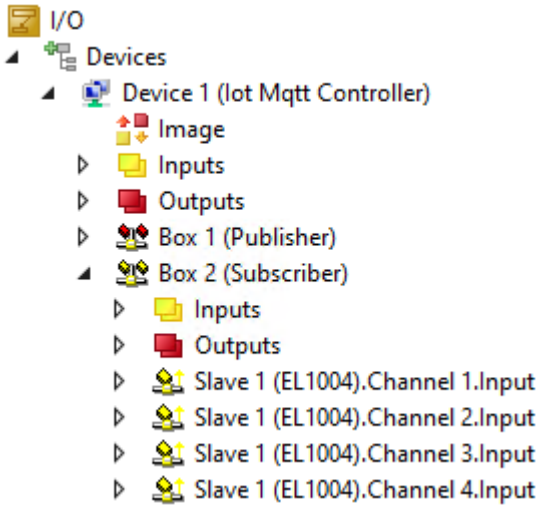




Anschließend können über einen Scan-Mechanismus die Symbolinformationen ausgelesen und automatisch entsprechende Ein-/Ausgangsvariablen im Prozessabbild des Geräts angelegt werden.



Resultat (exemplarisch am Beispiel des Subscribers):



Weitere Informationen

Nachdem die Konfiguration auf dem EK9160 aktiviert wurde, werden auf dem Message Broker drei Topics unterhalb des konfigurierten Main Topics verwendet:

1. Das Symbol-Topic beinhaltet die Symbolinformationen zu den angeschlossenen I/O Klemmen und wird vom EK9160 nach dem Herstellen der Kommunikationsverbindung mit dem Message Broker befüllt.
2. Das Description-Topic beinhaltet allgemeine Statusinformationen zum Gerät und wird vom EK9160 nach dem Herstellen der Kommunikationsverbindung mit dem Message Broker befüllt.
3. Das Daten-Topic beinhaltet die reinen Prozessdaten der angeschlossenen I/O Klemmen. Dieses Topic wird vom EK9160 somit zyklisch mit Daten befüllt.

Der folgende Screenshot zeigt einen Ausschnitt vom Mosquitto Message Broker im Verbose-Modus, auf welchem man die einzelnen Publishes des EK9160 auf die oben genannten Topics erkennt.

```
1632306820: Received PUBLISH from Device1_72ce5805_000010a49f2 (d0, q1, r1, m4, 'EK9160/EK-58CE72/Stream1/Bin/Tx/Symbols', ... (488 bytes
1632306820: Sending PUBACK to Device1_72ce5805_000010a49f2 (m4, rc0)
1632306820: Received PUBLISH from Device1_72ce5805_000010a49f2 (d0, q1, r1, m5, 'EK9160/EK-58CE72/Stream1/Bin/Tx/Desc', ... (179 bytes))
1632306820: Sending PUBACK to Device1_72ce5805_000010a49f2 (m5, rc0)
1632306821: Received PUBLISH from Device1_72ce5805_000010a49f2 (d0, q0, r0, m0, 'EK9160/EK-58CE72/Stream1/Bin/Tx/Data', ... (44 bytes))
1632306822: Received PUBLISH from Device1_72ce5805_000010a49f2 (d0, q0, r0, m0, 'EK9160/EK-58CE72/Stream1/Bin/Tx/Data', ... (44 bytes))
1632306823: Received PUBLISH from Device1_72ce5805_000010a49f2 (d0, q0, r0, m0, 'EK9160/EK-58CE72/Stream1/Bin/Tx/Data', ... (44 bytes))
1632306824: Received PUBLISH from Device1_72ce5805_000010a49f2 (d0, q0, r0, m0, 'EK9160/EK-58CE72/Stream1/Bin/Tx/Data', ... (44 bytes))
1632306825: Received PINGREQ from Device1_72ce5805_000010a49f2
1632306825: Sending PINGRESP to Device1_72ce5805_000010a49f2
1632306825: Received PUBLISH from Device1_72ce5805_000010a49f2 (d0, q0, r0, m0, 'EK9160/EK-58CE72/Stream1/Bin/Tx/Data', ... (44 bytes))
1632306826: Received PUBLISH from Device1_72ce5805_000010a49f2 (d0, q0, r0, m0, 'EK9160/EK-58CE72/Stream1/Bin/Tx/Data', ... (44 bytes))
1632306827: Received PUBLISH from Device1_72ce5805_000010a49f2 (d0, q0, r0, m0, 'EK9160/EK-58CE72/Stream1/Bin/Tx/Data', ... (44 bytes))
1632306828: Received PUBLISH from Device1_72ce5805_000010a49f2 (d0, q0, r0, m0, 'EK9160/EK-58CE72/Stream1/Bin/Tx/Data', ... (44 bytes))
1632306829: Received PUBLISH from Device1_72ce5805_000010a49f2 (d0, q0, r0, m0, 'EK9160/EK-58CE72/Stream1/Bin/Tx/Data', ... (44 bytes))
1632306830: Received PUBLISH from Device1_72ce5805_000010a49f2 (d0, q0, r0, m0, 'EK9160/EK-58CE72/Stream1/Bin/Tx/Data', ... (44 bytes))
1632306831: Received PUBLISH from Device1_72ce5805_000010a49f2 (d0, q0, r0, m0, 'EK9160/EK-58CE72/Stream1/Bin/Tx/Data', ... (44 bytes))
1632306832: Received PUBLISH from Device1_72ce5805_000010a49f2 (d0, q0, r0, m0, 'EK9160/EK-58CE72/Stream1/Bin/Tx/Data', ... (44 bytes))
1632306833: Received PUBLISH from Device1_72ce5805_000010a49f2 (d0, q0, r0, m0, 'EK9160/EK-58CE72/Stream1/Bin/Tx/Data', ... (44 bytes))
1632306834: Received PUBLISH from Device1_72ce5805_000010a49f2 (d0, q0, r0, m0, 'EK9160/EK-58CE72/Stream1/Bin/Tx/Data', ... (44 bytes))
```

4.9 Exponential backoff

Um im Falle eines Verbindungsfehlers mit dem Message Broker diesen nicht unnötig mit Verbindungsanforderungen zu belasten, kann auf eine Funktionalität namens "exponential backoff" zurückgegriffen werden. Hierbei wird nach einem TLS-Verbindungsfehler mit dem Message Broker die Reconnect-Rate multiplikativ angepasst. Diese Funktion ist über die Methode ActivateExponentialBackoff() [▶ 63] aktivierbar. Die Parameter der Methode geben hierbei die Minimal- und Maximalzeit für den Algorithmus an. Die Minimalzeit beschreibt den Anfangsverzögerungswert für den erneuten Verbindungsversuch. Die Maximalzeit beschreibt den größten Verzögerungswert. Die Verzögerungswerte werden bis zum Erreichen des Maximalwerts verdoppelt. Sobald eine Verbindung hergestellt werden konnte, wird die Backoff-Rate wieder auf den Ursprungswert zurückgesetzt. Über die Methode DeactivateExponentialBackoff() [▶ 64] kann diese Funktion auch programmatisch wieder außer Kraft gesetzt werden.

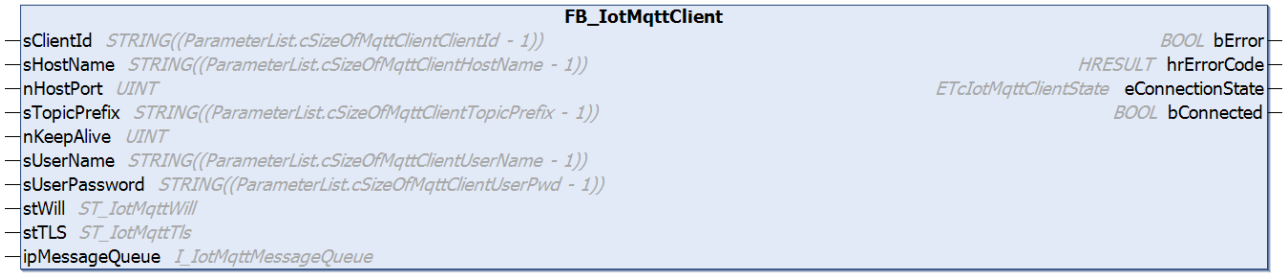
5 SPS API

5.1 Tc3_IotBase

5.1.1 MQTT3

Die folgenden Bausteine und Strukturen sind für MQTT in der Version 3.1.1 implementiert. Die Implementierungen für MQTT in der Version 5 finden sich unter MQTT5 [▶ 71].

5.1.1.1 FB_IotMqttClient



Der Funktionsbaustein ermöglicht die Kommunikation zu einem Message Broker basierend auf MQTT Version 3.1.1.

Ein Client-Funktionsbaustein ist hierbei für die Verbindung zu genau einem Broker zuständig. Um die Hintergrundkommunikation zu diesem Broker zu gewährleisten und Nachrichten empfangen zu können, muss die Methode Execute() [▶ 60] des Funktionsbausteins zyklisch aufgerufen werden.

Alle Verbindungsparameter sind als Eingangsparameter vorhanden und werden beim Verbindungsaufbau ausgewertet.

Syntax

Definition:

```
FUNCTION_BLOCK FB_IotMqttClient
VAR_INPUT
    sClientId        : STRING(255);     // default is generated during initialization
    sHostName        : STRING(255) := '127.0.0.1'; // default is local host
    nHostPort        : UINT := 1883;    // default is 1883
    sTopicPrefix     : STRING(255);     // topic prefix for pub and sub of this client (handled inte
rnally)
    nKeepAlive       : UINT := 60;      // in seconds

    sUserName        : STRING(255);     // optional parameter
    sUserPassword    : STRING(255);     // optional parameter
    stWill           : ST_IotMqttWill;  // optional parameter
    stTLS            : ST_IotMqttTls;   // optional parameter

    ipMessageQueue   : I_IotMqttMessageQueue; // if received messages should be queued during call o
f Execute()
END_VAR
VAR_OUTPUT
    bError           : BOOL;
    hrErrorCode      : HRESULT;
    eConnectionState : ETcIotMqttClientState;
    bConnected       : BOOL; // TRUE if connection to host is established
END_VAR
```

Eingänge

Name	Typ	Beschreibung
sClientId	STRING(255)	Die Client-ID kann individuell angegeben werden und muss bei den meisten Message Brokern eindeutig sein. Bei fehlender Angabe wird eine ID automatisch vom TwinCAT Treiber generiert, welche auf dem SPS Projektnamen basiert.
sHostName	STRING(255)	sHostName kann als Name oder als IP-Adresse angegeben werden. Bei fehlender Angabe wird Local Host verwendet.
nHostPort	UINT	Hier wird der Host Port angegeben. Dieser ist standardmäßig 1883.
sTopicPrefix	STRING(255)	Hier kann ein Topic Prefix angegeben werden, das automatisch für alle Publish- und Subscribe-Kommandos angefügt wird.
nKeepAlive	UINT	Hier kann die Watchdog-Zeit in Sekunden angegeben werden, mit der die Verbindung zwischen Client und Broker überwacht wird.
sUserName	STRING(255)	Optional kann ein Benutzername angegeben werden.
sUserPassword	STRING(255)	Zum Benutzernamen kann hier ein Passwort angegeben werden.
stWill	ST_IotMqttWill [▶ 65]	Bei irregulärem Verbindungsabbau des Clients vom Broker kann optional eine letzte vorkonfigurierte Nachricht vom Broker an das sogenannte „Will-Topic“ versendet werden. Hier wird diese Nachricht spezifiziert.
stTLS	ST_IotMqttTls [▶ 65]	Wenn der Broker eine TLS gesicherte Verbindung anbietet, kann hier die nötige Konfiguration vorgenommen werden.
ipMessageQueue	I_IotMqttMessageQueue	Hier kann optional eine Instanz von FB_IotMqttMessageQueue [▶ 66] zugewiesen werden. Wenn Sie eingehende neue Nachrichten in der Message Queue einsammeln wollen, ohne selbst die Callback-Methode zu implementieren, kann dieser Eingangsparameter gesetzt werden. (Siehe auch IotMqttSampleUsingQueue [▶ 319]) Wenn Sie eingehende neue Nachrichten ohne Message Queue auswerten wollen indem Sie die Callback-Methode selbst implementieren, wird dieser Eingangsparameter nicht benötigt. (Siehe auch IotMqttSampleUsingCallback [▶ 321]) Wenn die Callback-Methode verwendet bzw. überschrieben wird, wird keine Message Queue verwendet, unabhängig davon, ob ipMessageQueue gesetzt wurde.

Ausgänge

Name	Typ	Beschreibung
bError	BOOL	Wird TRUE, sobald eine Fehlersituation eintritt.
hrErrorCode	HRESULT	Liefert bei einem gesetzten bError-Ausgang einen Fehlercode. Eine Erläuterung zu den möglichen Fehlercodes befindet sich im Anhang.
eConnectionState	ETcIotMqttClientState [▶ 115]	Gibt den Zustand der Verbindung vom Client zum Broker als Enumeration ETcIotMqttClientState an.
bConnected	BOOL	Dieser Ausgang ist TRUE, wenn eine Verbindung vom Client zum Broker besteht.

Methoden

Name	Beschreibung
Execute [▶ 60]	Methode zur Hintergrundkommunikation mit dem TwinCAT-Treiber. Die Methode muss zyklisch aufgerufen werden.
Publish [▶ 60]	Methode zum Ausführen eines Publish an einen MQTT Message Broker.
Subscribe [▶ 61]	Methode zum Einrichten einer Subscription.
Unsubscribe [▶ 62]	Methode zum Entfernen einer Subscription.
ActivateExponentialBackoff [▶ 63]	Aktiviert die Funktion des exponential backoff [▶ 56]
DeactivateExponentialBackoff [▶ 64]	Deaktiviert die Funktion des exponential backoff [▶ 56]
GetTimeSinceLastBrokerMessage [▶ 64]	Methode zur Abfrage der Zeit (in ms) seit der letzten Nachricht vom Message Broker.

Ereignisgesteuerte Methoden (Callback-Methoden)

Name	Beschreibung
OnMqttMessage [▶ 62]	Callback-Methode, die vom TwinCAT-Treiber aufgerufen wird, wenn eine Subscription auf ein Topic erfolgt ist und eingehende Nachrichten empfangen werden.

Message-Payload-Formatierung

Beachten Sie, dass der Datentyp und die Formatierung des Inhalts der Sender- und Empfängerseite bekannt sein müssen, insbesondere beim Versand von Binärinformationen (Alignment) oder Strings (mit/ohne Nullterminierung).

Strings im UTF-8-Format

Die hier verwendeten Variablen vom Typ STRING nutzen das UTF-8-Format. Diese STRING-Formatierung ist üblich bei IoT/MQTT-Kommunikation sowie JSON-Dokumenten.

Um Sonderzeichen und Texte verschiedenster Sprachen empfangen zu können, wird der Zeichensatz in den Bibliotheken Tc3_IotBase und Tc3_JsonXml nicht auf den typischen Zeichensatz vom Datentyp STRING beschränkt. Stattdessen wird der Unicode-Zeichensatz als UTF-8-Format in Verbindung mit dem Datentyp STRING verwendet.

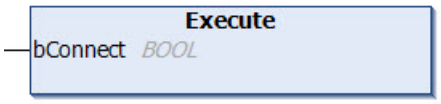
Bei Verwendung des ASCII-Zeichensatzes besteht kein Unterschied zwischen der typischen Formatierung in einem STRING und der UTF-8-Formatierung eines STRING.

Weitere Informationen zum UTF-8-STRING-Format sowie vorhandenen Anzeige- und Konvertierungsmöglichkeiten finden Sie in der Dokumentation der SPS-Bibliothek Tc2_Utilities.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.0	IPC oder CX (x86, x64, ARM)	Tc3_IotBase

5.1.1.1.1 Execute



Diese Methode muss zyklisch aufgerufen werden, um die Hintergrundkommunikation mit dem MQTT Broker zu gewährleisten.

Syntax

```
METHOD Execute
VAR_INPUT
    bConnect : BOOL;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
bConnect	BOOL	Wenn bConnect auf TRUE gesetzt wird, findet der Verbindungsaufbau zum Broker statt. Um die Verbindung zu halten, muss bConnect gesetzt bleiben.

Mögliche Fehler werden an den Ausgängen bError, hrErrorCode und eConnectionState der Bausteininstanz ausgegeben.

5.1.1.1.2 Publish

Diese Methode wird einmalig aufgerufen, um eine Nachricht zum Broker zu versenden.

Syntax

```
METHOD Publish : BOOL
VAR_IN_OUT
    sTopic       : STRING; // topic string (UTF-8) with any length (attend that MQTT topics are case
 sensitive)
END_VAR
VAR_INPUT
    pPayload     : PVOID;
    nPayloadSize : UDINT;
    eQoS         : TcIotMqttQos; // quality of service between the publishing client and the broker
    bRetain      : BOOL; // if TRUE the broker stores the message in order to send it to new subscri
bers
    bQueue       : BOOL; // for future extension
END_VAR
```

Rückgabewert

Name	Typ	Beschreibung
Publish	BOOL	Bei erfolgreichem Aufruf liefert die Methode den Rückgabewert TRUE.

Eingänge

Name	Typ	Beschreibung
sTopic	STRING	Topic der MQTT-Nachricht
pPayload	PVOID	Adresse zum Payload der MQTT-Nachricht
nPayloadSize	UDINT	Größe vom Payload in Bytes
eQoS	TcIotMqttQos	"Quality of Service"
bRetain	BOOL	Wenn bRetain TRUE ist, speichert der Broker die Nachricht, um sie später hinzukommenden Subscribern nachträglich zukommen zu lassen.
bQueue	BOOL	Reservierter Parameter, welcher immer mit FALSE belegt werden kann.

Mögliche Fehler werden an den Ausgängen bError und hrErrorCode der Bausteininstanz ausgegeben.

Message-Payload-Formatierung

Beachten Sie, dass der Datentyp und die Formatierung des Inhalts der Sender- und Empfängerseite bekannt sein müssen, insbesondere beim Versand von Binärinformationen (Alignment) oder Strings (mit/ohne Nullterminierung).

Strings im UTF-8-Format

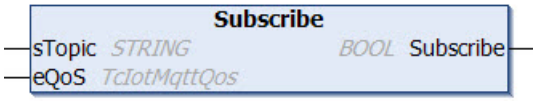
Die hier verwendeten Variablen vom Typ STRING nutzen das UTF-8-Format. Diese STRING-Formatierung ist üblich bei IoT/MQTT-Kommunikation sowie JSON-Dokumenten.

Um Sonderzeichen und Texte verschiedenster Sprachen empfangen zu können, wird der Zeichensatz in den Bibliotheken Tc3_IotBase und Tc3_JsonXml nicht auf den typischen Zeichensatz vom Datentyp STRING beschränkt. Stattdessen wird der Unicode-Zeichensatz als UTF-8-Format in Verbindung mit dem Datentyp STRING verwendet.

Bei Verwendung des ASCII-Zeichensatzes besteht kein Unterschied zwischen der typischen Formatierung in einem STRING und der UTF-8-Formatierung eines STRING.

Weitere Informationen zum UTF-8-STRING-Format sowie vorhandenen Anzeige- und Konvertierungsmöglichkeiten finden Sie in der Dokumentation der SPS-Bibliothek Tc2_Utilities.

5.1.1.1.3 Subscribe



Mit dieser Methode meldet der Client dem Broker, dass er alle MQTT-Nachrichten mit einem bestimmten Topic erhalten möchte. Die Methode kann eine MQTT-Client-Instanz auf mehrere Topics anmelden.

Syntax

```
METHOD Subscribe : BOOL
VAR_IN_OUT
    sTopic       : STRING; // topic string (UTF-8) with any length (attend that MQTT topics are case
 sensitive)
END_VAR
VAR_INPUT
    eQoS         : TcIotMqttQos; // quality of service between the publishing client and the broker
END_VAR
```

Rückgabewert

Name	Typ	Beschreibung
Subscribe	BOOL	Bei erfolgreichem Aufruf liefert die Methode den Rückgabewert TRUE.

Eingänge

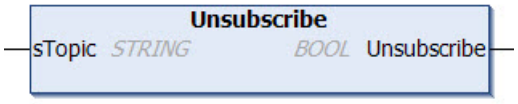
Name	Typ	Beschreibung
eQoS	TcIotMqttQos	"Quality of Service"

Ein-/Ausgänge

Name	Typ	Beschreibung
sTopic	STRING	Topic der MQTT-Nachricht

Mögliche Fehler werden an den Ausgängen bError und hrErrorCode der Bausteininstanz ausgegeben.

5.1.1.1.4 Unsubscribe



Mit dieser Methode meldet der Client dem Broker, dass keine weiteren Nachrichten mit dem angegebenen Topic mehr zu ihm übertragen werden sollen.

Syntax

```
METHOD Unsubscribe : BOOL
VAR_IN_OUT
    sTopic       : STRING; // topic string (UTF-8) with any length (attend that MQTT topics are case
 sensitive)
END_VAR
```

Rückgabewert

Name	Typ	Beschreibung
Unsubscribe	BOOL	Bei erfolgreichem Aufruf liefert die Methode den Rückgabewert TRUE.

Ein-/Ausgänge

Name	Typ	Beschreibung
sTopic	STRING	Topic, mit dem keine weiteren Nachrichten empfangen werden sollen.

Mögliche Fehler werden an den Ausgängen bError und hrErrorCode der Bausteininstanz ausgegeben.

5.1.1.1.5 OnMqttMessage

Callback-Methode

Diese Methode darf nicht vom Anwender aufgerufen werden. Stattdessen kann vom Funktionsbaustein FB_IotMqttClient abgeleitet und diese Methode überschrieben werden. Während dem Aufruf der Methode Execute()hat der zuständige TwinCAT-Treiber die Möglichkeit, im Fall von neuen eingehenden Nachrichten, die Methode OnMqttMessage() aufzurufen. Bei mehreren eingehenden Nachrichten wird die Callback-

Methode mehrfach, je Nachricht einmal, aufgerufen. Die Implementierung der Methode muss dies berücksichtigen.
Die Verwendung der Callback-Methode ist auch im Beispiel IotMqttSampleUsingCallback [▶ 321] erläutert.

Syntax

```
METHOD OnMqttMessage : HRESULT
VAR_IN_OUT CONSTANT
    topic    : STRING;
END_VAR
VAR_INPUT
    payload  : PVOID;
    length   : UDINT;
    qos      : TcIotMqttQos;
    repeated : BOOL;
END_VAR
```

Rückgabewert

Name	Typ	Beschreibung
OnMqttMessage	HRESULT	Der Rückgabewert der Methode ist mit S_OK zu belegen, sofern die Nachricht angenommen wurde. Soll die Nachricht im Kontext des nächsten Execute()-Aufrufs erneut ausgegeben werden, kann der Rückgabewert mit S_FALSE belegt werden.

Eingänge

Name	Typ	Beschreibung
payload	PVOID	Adresse des Payload der eingegangenen MQTT-Nachricht
length	UDINT	Größe vom Payload in Bytes
qos	TcIotMqttQos	"Quality of Service"
repeated	BOOL	Wurde der letzte OnMqttMessage()-Methodenaufruf vom Anwender nicht mit S_OK erwidert, wird die Nachricht im Kontext des nächsten Execute()-Aufrufs erneut ausgegeben und hierbei der Parameter repeated gesetzt. Daran ist zu erkennen, dass die Nachricht wiederholt ausgegeben wird.

Ein-/Ausgänge

Name	Typ	Beschreibung
topic	STRING	Topic der eingegangenen MQTT-Nachricht

5.1.1.1.6 ActivateExponentialBackoff

Um im Falle eines Verbindungsfehlers mit dem Message Broker diesen nicht unnötig mit Verbindungsanforderungen zu belasten, kann auf eine Funktionalität namens "exponential backoff" zurückgegriffen werden. Hierbei wird nach einem TLS-Verbindungsfehler mit dem Message Broker die Reconnect-Rate multiplikativ angepasst. Diese Funktion ist über die Methode ActivateExponentialBackoff() [▶ 63] aktivierbar. Die Parameter der Methode geben hierbei die Minimal- und Maximalzeit für den Algorithmus an. Die Minimalzeit beschreibt den Anfangsverzögerungswert für den erneuten Verbindungsversuch. Die Maximalzeit beschreibt den größten Verzögerungswert. Die Verzögerungswerte werden bis zum Erreichen des Maximalwerts verdoppelt. Sobald eine Verbindung hergestellt werden konnte, wird die Backoff-Rate wieder auf den Ursprungswert zurückgesetzt. Über die Methode DeactivateExponentialBackoff() [▶ 64] kann diese Funktion auch programmatisch wieder außer Kraft gesetzt werden.

Syntax

```
METHOD ActivateExponentialBackoff
VAR_INPUT
    tMqttBackoffMinTime: TIME;
    tMqttBackoffMaxTime: TIME;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
tMqttBackoffMinTime	TIME	Beschreibt den Anfangsverzögerungswert für den erneuten Verbindungsversuch.
tMqttBackoffMaxTime	TIME	Beschreibt den größten Verzögerungswert. Nach Erreichen dieses Werts, erfolgen alle erneuten Verbindungsversuche immer in diesen Abständen.

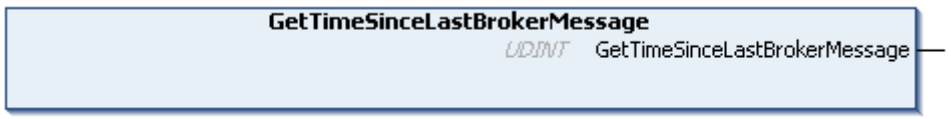
5.1.1.1.7 DeactivateExponentialBackoff

Um im Falle eines Verbindungsfehlers mit dem Message Broker diesen nicht unnötig mit Verbindungsanforderungen zu belasten, kann auf eine Funktionalität namens "exponential backoff" zurückgegriffen werden. Hierbei wird nach einem TLS-Verbindungsfehler mit dem Message Broker die Reconnect-Rate multiplikativ angepasst. Diese Funktion ist über die Methode ActivateExponentialBackoff() [▶ 63] aktivierbar. Die Parameter der Methode geben hierbei die Minimal- und Maximalzeit für den Algorithmus an. Die Minimalzeit beschreibt den Anfangsverzögerungswert für den erneuten Verbindungsversuch. Die Maximalzeit beschreibt den größten Verzögerungswert. Die Verzögerungswerte werden bis zum Erreichen des Maximalwerts verdoppelt. Sobald eine Verbindung hergestellt werden konnte, wird die Backoff-Rate wieder auf den Ursprungswert zurückgesetzt. Über die Methode DeactivateExponentialBackoff() [▶ 64] kann diese Funktion auch programmatisch wieder außer Kraft gesetzt werden.

Syntax

```
METHOD DeactivateExponentialBackoff
```

5.1.1.1.8 GetTimeSinceLastBrokerMessage



Diese Methode gibt die Zeit (in ms) seit der letzten Nachricht vom Message Broker an. Bei jeder neuen Nachricht vom Message Broker wird die Zeit wieder auf 0 zurückgesetzt. Hierbei ist es egal, ob es sich um Ping-Nachrichten oder reguläre Publish/Subscribe-Nachrichten handelt.

In der MQTT-Spezifikation ist definiert, dass ein MQTT-Client selbst bestimmt nach welcher Zeit er einen Timeout in Richtung des Message Brokers annimmt. Das ist mit dieser Methode aus der SPS möglich. Die Keep Alive-Zeit spezifiziert hingegen nach welcher Zeit (mit 1,5 multipliziert) der Message Broker bei ausbleibenden Nachrichten vom Client einen Timeout des Clients annimmt.

Syntax

```
METHOD GetTimeSinceLastBrokerMessage : UDINT
```

Rückgabewert

Name	Typ	Beschreibung
GetTimeSinceLastBrokerMessage	UDINT	Die Zeit in Millisekunden seit der letzten Nachricht vom Message Broker.

5.1.1.2 ST_IotMqttWill

Mittels folgender Angaben kann eine Nachricht spezifiziert werden, die bei einem irregulärem Verbindungsabbau zwischen Client und Broker als letzte Nachricht vom Broker zu den Subscribern versendet wird, die das entsprechende Topic abonniert haben.

Syntax

Definition:

```
TYPE ST_IotMqttWill :
STRUCT
    {attribute 'TcEncoding':='UTF-8'}
    sTopic        : STRING(255); // topic string (UTF-8) (attend that MQTT topics are case sensitive
)
    pPayload      : PVOID;
    nPayloadSize  : UDINT;
    eQoS          : TcIotMqttQos := TcIotMqttQos.ExactlyOnceDelivery; // quality of service between
the publishing client and the broker
    bRetain       : BOOL; // if TRUE the broker stores the message in order to send it to new subscr
ibers
END_STRUCT
END_TYPE
```

Parameter

Name	Typ	Beschreibung
sTopic	STRING(255)	Topic der Nachricht
pPayload	PVOID	Adresse zum Payload der Nachricht
nPayloadSize	UDINT	Größe in Bytes vom Payload
eQoS	TcIotMqttQos	Die „Quality of Service“ bietet folgende Einstellmöglichkeiten: QoS-Level 0, QoS-Level 1, QoS-Level 2 (Siehe QoS)
bRetain	BOOL	Wenn bRetain TRUE ist, speichert der Broker die Nachricht, um sie später hinzukommenden Subscribern nachträglich zukommen zu lassen.

Message-Payload-Formatierung

Beachten Sie, dass der Datentyp und die Formatierung des Inhalts der Sender- und Empfängerseite bekannt sein müssen, insbesondere beim Versand von Binärinformationen (Alignment) oder Strings (mit/ohne Nullterminierung).

5.1.1.3 ST_IotMqttTLS

TLS-Sicherheitseinstellung für den MQTT Client.

Es kann entweder CA (certificate authority) oder PSK verwendet werden.

Syntax

Definition:

```
TYPE ST_IotMqttTls :
STRUCT
    sCA                : STRING(255); // certificate authority as filename (PEM or DER format) or as
 string (PEM)
    sCAPath            : STRING(255); // for future use
    sCert              : STRING(255); // client certificate as filename (PEM or DER format) or as st
ring (PEM)
    sKeyFile           : STRING(255);
    sKeyPwd            : STRING(255);
    sCrl               : STRING(255); // Certificate Revocation List as filename (PEM or DER format)
 or as string (PEM)
    sCiphers           : STRING(255);
    sVersion           : STRING(80) := 'tlsv1.2'; // TLS version
    bNoServerCertCheck : BOOL := FALSE;

    sPskIdentity       : STRING(255);
    aPskKey            : ARRAY[1..64] OF BYTE;
    nPskKeyLen         : USINT;
```

```
    sAzureSas           : STRING(511);
END_STRUCT
END_TYPE
```

Parameter

Name	Typ	Beschreibung
sCA	STRING(255)	Zertifikat der Certificate Authority (CA)
sCert	STRING(255)	Client-Zertifikat, welches zur Authentifizierung am Broker verwendet werden soll
sKeyFile	STRING(255)	Privater Schlüssel des Clients
sKeyPwd	STRING(255)	Passwort des privaten Schlüssels, falls anwendbar
sCrl	STRING(255)	Pfad zur Certificate Revocation List, welche im Format PEM oder DER vorliegen kann
sCiphers	STRING(255)	Zu verwendende Cipher Suites, angegeben im OpenSSL-String-Format
sVersion	STRING(80)	Zu verwendende TLS-Version
bNoServerCertCheck	BOOL	Deaktiviert die Überprüfung des Server-Zertifikats auf Gültigkeit. Wenn ohne TLS-Verschlüsselung kommuniziert werden soll, muss dieser Wert auf FALSE bleiben.
sPskIdentity	STRING(255)	PreSharedKey-Identität für TLS-PSK Verbindung
aPskKey	ARRAY[1..64] OF BYTE	PreSharedKey für TLS-PSK-Verbindung
nPskKeyLen	USINT	Länge des PreSharedKey in Bytes
sAzureSAS	STRING(511)	SAS Token für Verbindung mit dem Microsoft Azure IoT Hub

5.1.1.4 FB_IotMqttMessageQueue

Dieser Funktionsbaustein bietet eine Message Queue für MQTT-Nachrichten, welche mit dem Baustein FB_IotMqttClient [▶ 57] verwendet werden kann. Hierzu wird eine Instanz deklariert und dem Eingang von FB_IotMqttClient übergeben. Der Funktionsbaustein arbeitet nach dem First-in-First-out-Prinzip (FiFo). Es ist möglich, dass mehrere MQTT-Nachrichten innerhalb eines SPS Zyklus empfangen und an der Message Queue bereit gestellt werden.

Im Programmablauf kann mit dem Property `nQueuedMessages` geprüft werden, ob und wie viele Nachrichten in der Message Queue eingesammelt wurden. Diese Nachrichten werden mit der Methode `Dequeue()` aus dem FiFo entnommen. Hierbei wird demnach die älteste Nachricht zuerst ausgegeben.

Größe der MQTT Message Queue

Die Menge der maximal möglichen Nachrichten in der Queue lässt sich über den Paramater `cMaxEntriesInMqttMessageQueue` in der Parameterliste [▶ 117] der Bibliothek Tc3_IotBase einstellen. Standardmäßig sind dies 1000 Nachrichten. Weil eine zeitnahe Verarbeitung der Nachrichten in den allermeisten Fällen verlangt wird, ist eine Anpassung meist nicht notwendig.

Die MQTT Message Queue allokiert für neue Nachrichten entsprechend der Topic- und Payload-Größe neuen Speicher. Standardmäßig ist die maximale Größe einer Nachricht auf 100 KB und die Größe einer MQTT Message Queue auf 1000 KB begrenzt. Dies kann für Sonderfälle ebenso in der Parameterliste [▶ 117] angepasst werden.

Eigenschaften

Name	Typ	Zugriff	Beschreibung
bOverwriteOldestEntry	BOOL	Get, Set	Hier kann parametriert werden, ob, wenn die Warteschlange voll ist, eine neu empfangene Nachricht die älteste Nachricht überschreiben soll. Wenn ja (TRUE), geht die älteste Nachricht verloren. Wenn nein (FALSE), geht die neueste Nachricht verloren. Eine volle Warteschlange ist ein unwahrscheinlicher Fall, wenn der Anwender für ein zügiges Auslesen der Queue mittels `Dequeue()` Methode sorgt.
nQueuedMessages	UDINT	Get	Gibt die Anzahl der aktuell in der Queue gesammelten MQTT-Nachrichten aus.

Methoden

Name	Beschreibung
Dequeue() [▶ 67]	Entnimmt der Warteschlange eine MQTT-Nachricht.
ResetQueue() [▶ 68]	Löscht alle Nachrichten aus der Warteschlange.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.0	IPC oder CX (x86, x64, ARM)	Tc3_IotBase

5.1.1.4.1 Dequeue

Wenn die Entnahme einer MQTT-Nachricht aus der Queue erfolgreich war, gibt die Methode den Rückgabewert TRUE aus. Die Menge der aktuell in der Queue gesammelten MQTT-Nachrichten (Property nQueuedMessages) wird durch Entnahme einer Nachricht um eins reduziert.

Syntax

```
METHOD Dequeue : BOOL
VAR_INPUT
    fbMessage : REFERENCE TO FB_IotMqttMessage;
END_VAR
```

Rückgabewert

Name	Typ	Beschreibung
Dequeue	BOOL	Bei erfolgreichem Aufruf liefert die Methode den Rückgabewert TRUE.

Eingänge

Name	Typ	Beschreibung
fbMessage	REFERENCE TO FB_IotMqttMessage	Für die Nachricht selbst wird die Referenz auf eine Instanz vom Typ FB_IotMqttMessage [▶ 68] übergeben.

5.1.1.4.2 ResetQueue

Mit Aufruf der Methode werden alle gesammelten MQTT-Nachrichten aus der Queue gelöscht.

Syntax

```
METHOD ResetQueue : BOOL
```

Rückgabewert

Name	Typ	Beschreibung
ResetQueue	BOOL	Bei erfolgreichem Aufruf liefert die Methode den Rückgabewert TRUE.

5.1.1.5 FB_IotMqttMessage

Wenn der TwintCAT MQTT Client (FB_IotMqttClient [▶ 57]) in Kombination mit einer Message Queue (FB_IotMqttMessageQueue [▶ 66]) verwendet wird, wird eine MQTT-Nachricht vom Baustein FB_IotMqttMessage repräsentiert. Ankommende Nachrichten werden von der Message Queue gesammelt und dem Anwender in Form einer solchen Bausteininstanz zur Verfügung gestellt.

Die Topic- und die Payload-Größe sowie der Parameter „Quality of Service“ der MQTT-Nachricht sind sofort am Bausteinausgang als Properties verfügbar. Das Topic selbst sowie auch das Payload kann mittels der bereitgestellten Methoden CompareTopic(), GetTopic() und GetPayload() einfach ausgewertet oder kopiert werden.

Message-Payload-Formatierung

Beachten Sie, dass der Datentyp und die Formatierung des Inhalts der Sender- und Empfängerseite bekannt sein müssen, insbesondere beim Versand von Binärinformationen (Alignment) oder Strings (mit/ohne Nullterminierung).

Größe einer MQTT-Nachricht

Die maximale Größe einer in der SPS zu empfangenden MQTT-Nachricht ist von der Hardware-Plattform abhängig und sollte selbst auf leistungsstärkeren/größeren Plattformen einige wenige MB nicht überschreiten.

Die maximal mögliche Größe einer Nachricht lässt sich über den Parameter `cMaxSizeOfMqttMessage` in der Parameterliste [▶ 117] der Bibliothek Tc3_IotBase einstellen. Standardmäßig ist die Größe einer Nachricht auf 100 KB begrenzt.

Bei Verwendung der MQTT Message Queue allokiert diese für neue Nachrichten dynamisch neuen Speicher aus dem TwinCAT Router Speicher.

Eigenschaften

Name	Typ	Zugriff	Beschreibung
eQoS	TcIotMqttQos	Get	„Quality of Service“ der MQTT-Nachricht
nPayloadSize	UDINT	Get	Größe vom Payload in Bytes
nTopicSize	UINT	Get	Größe vom Topic in Bytes

Methoden

Name	Beschreibung
CompareTopic() [▶ 69]	Überprüft ein angegebenes Topic mit dem Topic in der MQTT-Nachricht
GetTopic() [▶ 70]	Liefert das Topic einer MQTT-Nachricht
GetPayload() [▶ 70]	Liefert den Inhalt einer MQTT-Nachricht

Strings im UTF-8-Format

Die hier verwendeten Variablen vom Typ STRING nutzen das UTF-8-Format. Diese STRING-Formatierung ist üblich bei IoT/MQTT-Kommunikation sowie JSON-Dokumenten.

Um Sonderzeichen und Texte verschiedenster Sprachen empfangen zu können, wird der Zeichensatz in den Bibliotheken Tc3_IotBase und Tc3_JsonXml nicht auf den typischen Zeichensatz vom Datentyp STRING beschränkt. Stattdessen wird der Unicode-Zeichensatz als UTF-8-Format in Verbindung mit dem Datentyp STRING verwendet.

Bei Verwendung des ASCII-Zeichensatzes besteht kein Unterschied zwischen der typischen Formatierung in einem STRING und der UTF-8-Formatierung eines STRING.

Weitere Informationen zum UTF-8-STRING-Format sowie vorhandenen Anzeige- und Konvertierungsmöglichkeiten finden Sie in der Dokumentation der SPS-Bibliothek Tc2_Utilities.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.0	IPC oder CX (x86, x64, ARM)	Tc3_IotBase

5.1.1.5.1 CompareTopic

Die Methode liefert TRUE als Rückgabewert, wenn das angegebene Topic mit dem Topic der MQTT-Nachricht im Funktionsbaustein identisch ist.

Syntax

```
METHOD CompareTopic : BOOL
VAR_IN_OUT CONSTANT
    sTopic : STRING; // topic string with any length (attend that MQTT topics are case sensitive)
END_VAR
```

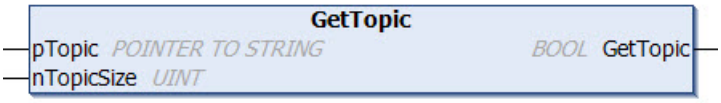
Rückgabewert

Name	Typ	Beschreibung
CompareTopic	BOOL	Ist TRUE, wenn das angegebene Topic mit dem Topic der MQTT-Nachricht im Funktionsbaustein identisch ist.

Ein-/Ausgänge

Name	Typ	Beschreibung
sTopic	STRING	

5.1.1.5.2 GetTopic



Syntax

```
METHOD GetTopic : BOOL
VAR_INPUT
    pTopic     : POINTER TO STRING; // topic buffer
    nTopicSize : UINT; // maximum size of topic buffer in bytes
END_VAR
```

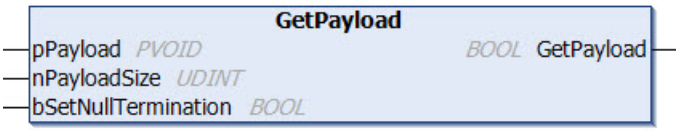
Rückgabewert

Name	Typ	Beschreibung
GetTopic	BOOL	

Eingänge

Name	Typ	Beschreibung
pTopic	POINTER TO STRING	Hier wird die Speicheradresse des Puffers angegeben, in den das Topic kopiert werden soll.
nTopicSize	UINT	Hier wird die maximal zur Verfügung stehende Größe in Bytes des Puffers angegeben.

5.1.1.5.3 GetPayload



Syntax

```
METHOD GetPayload : BOOL
VAR_INPUT
    pPayload           : PVOID; // payload buffer
    nPayloadSize       : UDINT; // maximum size of payload buffer in bytes
    bSetNullTermination: BOOL; // The publisher specifies the kind of payload. If it is a string, it
 could be null terminated or not. Setting this input to TRUE will force a null termination. One more
 byte is required for that.
END_VAR
```

Rückgabewert

Name	Typ	Beschreibung
GetPayload	BOOL	

Eingänge

Name	Typ	Beschreibung
pPayload	PVOID	Hier wird die Speicheradresse des Puffers, in den das Payload kopiert werden soll angegeben.
nPayloadSize	UDINT	Hier wird die maximal zur Verfügung stehende Größe in Bytes des Puffers angegeben.
bSetNullTermination	BOOL	Erfordert die Art des Payloads eine Null-Terminierung (String), so kann diese hiermit beim Kopiervorgang vorgenommen werden. Hat die Quelle der Nachricht (Publisher) bereits eine Null-Terminierung vorgenommen und diese in der Größenangabe des Payloads berücksichtigt, so ist dies nicht notwendig. Oft bestehen hierzu jedoch keine verlässlichen Informationen.

5.1.2 MQTT5

Die folgenden Bausteine und Strukturen sind für MQTT in der Version 5 implementiert.

5.1.2.1 FB_IotMqtt5Client

Der Funktionsbaustein ermöglicht die Kommunikation zu einem Message Broker basierend auf MQTT Version 5.0.

Ein Client-Funktionsbaustein ist hierbei für die Verbindung zu genau einem Broker zuständig. Um die Hintergrundkommunikation zu diesem Broker zu gewährleisten und Nachrichten empfangen zu können, muss die Methode Execute() des Funktionsbausteins zyklisch aufgerufen werden.

Alle Verbindungsparameter sind als Eingangsparameter vorhanden und werden beim Verbindungsaufbau ausgewertet.

Verwenden Sie den `FB_IotMqtt5Client`, wenn Sie keine Callback-Methoden selbst implementieren möchten und eingehende neue Nachrichten in der am Ausgang vorhandenen Message Queue bereitgestellt werden sollen. Verwenden Sie den FB_IotMqtt5ClientBase, wenn Sie die Callback-Methoden (`OnMqtt5Message()` und andere) selbst implementieren möchten.

Syntax

Definition:

```
FUNCTION_BLOCK FB_IotMqtt5Client
VAR_INPUT
    {attribute 'TcEncoding':='UTF-8'}
    sClientId        : STRING(255);        // default is generated during initialization
    {attribute 'TcEncoding':='UTF-8'}
    sHostName        : STRING(255) := '127.0.0.1'; // default is local host
    nHostPort        : UINT := 1883;       // default is 1883
    {attribute 'TcEncoding':='UTF-8'}
    sTopicPrefix     : STRING(255);        // topic prefix for pub and sub of this client (handled i
nternally)
    nKeepAlive       : UINT := 60;         // in seconds

    {attribute 'TcEncoding':='UTF-8'}
    sUserName        : STRING(255);        // optional parameter
    {attribute 'TcEncoding':='UTF-8'}
    sUserPassword    : STRING(255);        // optional parameter
    stWill           : ST_IotMqtt5Will;    // optional parameter
    stTLS            : ST_IotMqtt5Tls;     // optional parameter
    stAuth           : ST_IotMqtt5Auth;    // optional parameter
    stConnect        : ST_IotMqtt5Connect; // optional parameter
END_VAR
VAR_OUTPUT
    bError            : BOOL;
    hrErrorCode       : HRESULT;
    eConnectionState  : ETcIotMqttClientState;
    bConnected        : BOOL; // TRUE if connection to host is established
    fbMessageQueue    : FB_IotMqtt5MessageQueue;         // received messages are queued during call
of Execute()
```

```
    fbConnAckProps    : FB_IotMqtt5ConnAckProperties;    // info is set when a connection is
acknowledged
    fbDisconnectProps : FB_IotMqtt5DisconnectProperties; // info is set after disconnection
END_VAR
```

Eingänge

Name	Typ	Beschreibung
sClientId	STRING(255)	Die Client-ID kann individuell angegeben werden und muss bei den meisten Message Brokern eindeutig sein. Bei fehlender Angabe wird eine ID automatisch vom TwinCAT Treiber generiert, welche auf dem SPS Projektnamen basiert.
sHostName	STRING(255)	sHostName kann als Name oder als IP-Adresse angegeben werden. Bei fehlender Angabe wird Local Host verwendet.
nHostPort	UINT	Hier wird der Host Port angegeben. Dieser ist standardmäßig 1883.
sTopicPrefix	STRING(255)	Hier kann ein Topic Prefix angegeben werden, das automatisch für alle Publish- und Subscribe-Kommandos angefügt wird.
nKeepAlive	UINT	Hier kann die Watchdog-Zeit in Sekunden angegeben werden, mit der die Verbindung zwischen Client und Broker überwacht wird.
sUserName	STRING(255)	Optional kann ein Benutzername angegeben werden.
sUserPassword	STRING(255)	Zum Benutzernamen kann hier ein Passwort angegeben werden.
stWill	ST_IotMqtt5Will [▶ 94]	Bei irregulärem Verbindungsabbau des Clients vom Broker kann optional eine letzte vorkonfigurierte Nachricht vom Broker an das sogenannte „Will-Topic“ versendet werden. Hier wird diese Nachricht spezifiziert.
stTLS	ST_IotMqtt5Tls [▶ 95]	Wenn der Broker eine TLS gesicherte Verbindung anbietet, kann hier die nötige Konfiguration vorgenommen werden.
stAuth	ST_IotMqtt5Auth [▶ 96]	Mit dieser Struktur können erweiterte Authentifizierungsinformationen an den Message Broker übergeben werden.
stConnect	ST_IotMqtt5Connect [▶ 97]	Mit dieser Struktur können erweiterte Verbindungseinstellungen an den Message Broker übergeben werden.

Ausgänge

Name	Typ	Beschreibung
bError	BOOL	Wird TRUE, sobald eine Fehlersituation eintritt.
hrErrorCode	HRESULT	Liefert bei einem gesetzten bError-Ausgang einen Fehlercode. Eine Erläuterung zu den möglichen Fehlercodes befindet sich im Anhang.
eConnectionState	ETcIotMqttClientState [▶ 115]	Gibt den Zustand der Verbindung vom Client zum Broker als Enumeration ETcIotMqttClientState an.
bConnected	BOOL	Dieser Ausgang ist TRUE, wenn eine Verbindung vom Client zum Broker besteht.
fbMessageQueue	FB_IotMqtt5MessageQueue [▶ 98]	Die Message Queue der empfangenen Nachrichten wird am Ausgang bereitgestellt. Es kann direkt auf diesen Ausgang zugegriffen werden, um mit der Message Queue zu arbeiten.
fbConnAckProps	FB_IotMqtt5ConnAckProperties [▶ 105]	Wenn der Client vom Message Broker ConnAck Properties empfangen hat, werden diese hier angezeigt.
fbDisconnectProps	FB_IotMqtt5DisconnectProperties [▶ 109]	Wenn der Client vom Message Broker Disconnect Properties empfangen hat, werden diese hier angezeigt.

Methoden

Name	Beschreibung
Execute [▶ 74]	Methode zur Hintergrundkommunikation mit dem TwinCAT-Treiber. Die Methode muss zyklisch aufgerufen werden.
Publish [▶ 74]	Methode zum Ausführen eines Publish an einen MQTT Message Broker.
Request [▶ 78]	Hiermit kann ein Request im Rahmen des Request/Response Verfahrens von MQTTv5 verschickt werden.
Response [▶ 80]	Hiermit kann eine Response im Rahmen des Request/Response Verfahrens von MQTTv5 verschickt werden.
Subscribe [▶ 76]	Methode zum Einrichten einer Subscription.
Unsubscribe [▶ 76]	Methode zum Entfernen einer Subscription.
ActivateExponentialBackoff [▶ 77]	Aktiviert die Funktion des exponential backoff [▶ 56]
DeactivateExponentialBackoff [▶ 78]	Deaktiviert die Funktion des exponential backoff [▶ 56]
GetTimeSinceLastBrokerMessage [▶ 81]	Methode zur Abfrage der Zeit (in ms) seit der letzten Nachricht vom Message Broker.

Message-Payload-Formatierung

Beachten Sie, dass der Datentyp und die Formatierung des Inhalts der Sender- und Empfängerseite bekannt sein müssen, insbesondere beim Versand von Binärinformationen (Alignment) oder Strings (mit/ohne Nullterminierung).

Strings im UTF-8-Format

Die hier verwendeten Variablen vom Typ STRING nutzen das UTF-8-Format. Diese STRING-Formatierung ist üblich bei IoT/MQTT-Kommunikation sowie JSON-Dokumenten.

Um Sonderzeichen und Texte verschiedenster Sprachen empfangen zu können, wird der Zeichensatz in den Bibliotheken Tc3_IotBase und Tc3_JsonXml nicht auf den typischen Zeichensatz vom Datentyp STRING beschränkt. Stattdessen wird der Unicode-Zeichensatz als UTF-8-Format in Verbindung mit dem Datentyp STRING verwendet.

Bei Verwendung des ASCII-Zeichensatzes besteht kein Unterschied zwischen der typischen Formatierung in einem STRING und der UTF-8-Formatierung eines STRING.

Weitere Informationen zum UTF-8-STRING-Format sowie vorhandenen Anzeige- und Konvertierungsmöglichkeiten finden Sie in der Dokumentation der SPS-Bibliothek Tc2_Utilities.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4026.0	IPC oder CX (x86, x64, ARM)	Tc3_IotBase (>= v3.4.2.0)

5.1.2.1.1 Execute

Diese Methode muss zyklisch aufgerufen werden, um die Hintergrundkommunikation mit dem MQTT Broker zu gewährleisten.

Mögliche Fehler werden an den Ausgängen `bError` und `hrErrorCode` der Bausteininstanz ausgegeben.

Syntax

```
METHOD Execute
VAR_INPUT
    bConnect : BOOL;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
bConnect	BOOL	Wenn bConnect auf TRUE gesetzt wird, findet der Verbindungsaufbau zum Broker statt. Um die Verbindung zu halten, muss bConnect gesetzt bleiben.

5.1.2.1.2 Publish

Diese Methode wird einmalig aufgerufen, um eine Nachricht zum Broker zu versenden.

Mögliche Fehler werden an den Ausgängen `bError` und `hrErrorCode` der Bausteininstanz ausgegeben.

Syntax

```
METHOD Publish : BOOL
VAR_IN_OUT
    sTopic       : STRING; // topic string (UTF-8) with any length (attend that MQTT topics are case
 sensitive)
END_VAR
VAR_INPUT
    pPayload     : PVOID;
    nPayloadSize : UDINT;
    eQoS         : TcIotMqttQos; // quality of service between the publishing client and the broker
    bRetain      : BOOL; // if TRUE the broker stores the message in order to send it to new subscri
bers
    bQueue       : BOOL; // for future extension
    pProps       : POINTER TO MqttPublishProperties; // optional
END_VAR
```

Rückgabewert

Name	Typ	Beschreibung
Publish	BOOL	Bei erfolgreichem Aufruf liefert die Methode den Rückgabewert TRUE.

Eingänge

Name	Typ	Beschreibung
pPayload	PVOID	Adresse zum Payload der MQTT-Nachricht
nPayloadSize	UDINT	Größe vom Payload in Bytes
eQoS	TcIotMqttQos	"Quality of Service"
bRetain	BOOL	Wenn bRetain TRUE ist, speichert der Broker die Nachricht, um sie später hinzukommenden Subscribern nachträglich zukommen zu lassen.
bQueue	BOOL	Reservierter Parameter, welcher immer mit FALSE belegt werden kann.
pProps	POINTER TO MqttPublishProperties	Pointer auf zu verschickende Publish Properties. Der Parameter ist optional. Um Publish Properties anzugeben kann der FB_IotMqtt5PublishProperties [▶ 110] zur Hilfe genommen werden.

Ein-/Ausgänge

Name	Typ	Beschreibung
sTopic	STRING	Topic der MQTT-Nachricht

Message-Payload-Formatierung

Beachten Sie, dass der Datentyp und die Formatierung des Inhalts der Sender- und Empfängerseite bekannt sein müssen, insbesondere beim Versand von Binärinformationen (Alignment) oder Strings (mit/ohne Nullterminierung).

Strings im UTF-8-Format

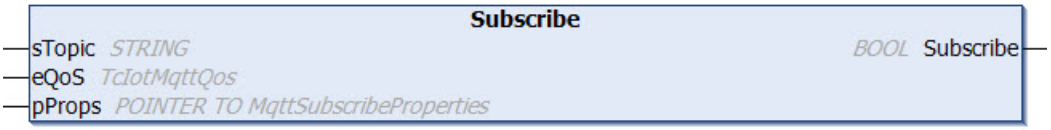
Die hier verwendeten Variablen vom Typ STRING nutzen das UTF-8-Format. Diese STRING-Formatierung ist üblich bei IoT/MQTT-Kommunikation sowie JSON-Dokumenten.

Um Sonderzeichen und Texte verschiedenster Sprachen empfangen zu können, wird der Zeichensatz in den Bibliotheken Tc3_IotBase und Tc3_JsonXml nicht auf den typischen Zeichensatz vom Datentyp STRING beschränkt. Stattdessen wird der Unicode-Zeichensatz als UTF-8-Format in Verbindung mit dem Datentyp STRING verwendet.

Bei Verwendung des ASCII-Zeichensatzes besteht kein Unterschied zwischen der typischen Formatierung in einem STRING und der UTF-8-Formatierung eines STRING.

Weitere Informationen zum UTF-8-STRING-Format sowie vorhandenen Anzeige- und Konvertierungsmöglichkeiten finden Sie in der Dokumentation der SPS-Bibliothek Tc2_Utilities.

5.1.2.1.3 Subscribe



Mit dieser Methode meldet der Client dem Broker, dass er alle MQTT-Nachrichten mit einem bestimmten Topic erhalten möchte. Die Methode kann eine MQTT-Client-Instanz auf mehrere Topics anmelden.

Mögliche Fehler werden an den Ausgängen `bError` und `hrErrorCode` der Bausteininstanz ausgegeben.

Syntax

```
METHOD Subscribe : BOOL
VAR_IN_OUT
    sTopic    : STRING; // topic string (UTF-8) with any length (attend that MQTT topics are case se
nsitive)
END_VAR
VAR_INPUT
    eQoS      : TcIotMqttQos; // quality of service between the publishing client and the broker
    pProps    : POINTER TO MqttSubscribeProperties; // optional
END_VAR
```

Rückgabewert

Name	Typ	Beschreibung
Subscribe	BOOL	Bei erfolgreichem Aufruf liefert die Methode den Rückgabewert TRUE.

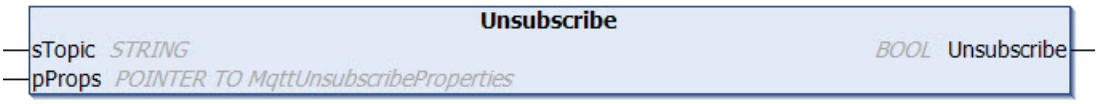
Eingänge

Name	Typ	Beschreibung
eQoS	TcIotMqttQos	"Quality of Service"
pProps	POINTER TO MqttSubscribeProperties	Pointer auf zu verschickende Subscribe Properties. Der Parameter ist optional. Um Subscribe Properties anzugeben kann der FB_IotMqtt5SubscribeProperties [▶ 112] zur Hilfe genommen werden.

Ein-/Ausgänge

Name	Typ	Beschreibung
sTopic	STRING	Topic der MQTT-Nachricht

5.1.2.1.4 Unsubscribe



Mit dieser Methode meldet der Client dem Broker, dass keine weiteren Nachrichten mit dem angegebenen Topic mehr zu ihm übertragen werden sollen.

Mögliche Fehler werden an den Ausgängen `bError` und `hrErrorCode` der Bausteininstanz ausgegeben.

Syntax

```
METHOD Unsubscribe : BOOL
VAR_IN_OUT
    sTopic    : STRING; // topic string (UTF-8) with any length (attend that MQTT topics are case se
nsitive)
```

```
END_VAR
VAR_INPUT
    pProps    : POINTER TO MqttUnsubscribeProperties; // optional
END_VAR
```

Rückgabewert

Name	Typ	Beschreibung
Unsubscribe	BOOL	Bei erfolgreichem Aufruf liefert die Methode den Rückgabewert TRUE.

Eingänge

Name	Typ	Beschreibung
pProps	POINTER TO MqttUnsubscribeProperties	Pointer auf zu verschickende Unsubscribe Properties. Der Parameter ist optional. Um Unsubscribe Properties anzugeben kann der FB_IotMqtt5UnsubscribeProperties [▶ 113] zur Hilfe genommen werden.

Ein-/Ausgänge

Name	Typ	Beschreibung
sTopic	STRING	Topic, mit dem keine weiteren Nachrichten empfangen werden sollen.

5.1.2.1.5 ActivateExponentialBackoff

Um im Falle eines Verbindungsfehlers mit dem Message Broker diesen nicht unnötig mit Verbindungsanforderungen zu belasten, kann auf eine Funktionalität namens "exponential backoff" zurückgegriffen werden. Hierbei wird nach einem TLS-Verbindungsfehler mit dem Message Broker die Reconnect-Rate multiplikativ angepasst. Diese Funktion ist über die Methode ActivateExponentialBackoff() [▶ 63] aktivierbar. Die Parameter der Methode geben hierbei die Minimal- und Maximalzeit für den Algorithmus an. Die Minimalzeit beschreibt den Anfangsverzögerungswert für den erneuten Verbindungsversuch. Die Maximalzeit beschreibt den größten Verzögerungswert. Die Verzögerungswerte werden bis zum Erreichen des Maximalwerts verdoppelt. Sobald eine Verbindung hergestellt werden konnte, wird die Backoff-Rate wieder auf den Ursprungswert zurückgesetzt. Über die Methode DeactivateExponentialBackoff() [▶ 64] kann diese Funktion auch programmatisch wieder außer Kraft gesetzt werden.

Syntax

```
METHOD ActivateExponentialBackoff
VAR_INPUT
    tMqttBackoffMinTime: TIME;
    tMqttBackoffMaxTime: TIME;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
tMqttBackoffMinTime	TIME	Beschreibt den Anfangsverzögerungswert für den erneuten Verbindungsversuch.
tMqttBackoffMaxTime	TIME	Beschreibt den größten Verzögerungswert. Nach Erreichen dieses Werts, erfolgen alle erneuten Verbindungsversuche immer in diesen Abständen.

5.1.2.1.6 DeactivateExponentialBackoff

Um im Falle eines Verbindungsfehlers mit dem Message Broker diesen nicht unnötig mit Verbindungsanforderungen zu belasten, kann auf eine Funktionalität namens "exponential backoff" zurückgegriffen werden. Hierbei wird nach einem TLS-Verbindungsfehler mit dem Message Broker die Reconnect-Rate multiplikativ angepasst. Diese Funktion ist über die Methode ActivateExponentialBackoff() [▶ 63] aktivierbar. Die Parameter der Methode geben hierbei die Minimal- und Maximalzeit für den Algorithmus an. Die Minimalzeit beschreibt den Anfangsverzögerungswert für den erneuten Verbindungsversuch. Die Maximalzeit beschreibt den größten Verzögerungswert. Die Verzögerungswerte werden bis zum Erreichen des Maximalwerts verdoppelt. Sobald eine Verbindung hergestellt werden konnte, wird die Backoff-Rate wieder auf den Ursprungswert zurückgesetzt. Über die Methode DeactivateExponentialBackoff() [▶ 64] kann diese Funktion auch programmatisch wieder außer Kraft gesetzt werden.

Syntax

```
METHOD DeactivateExponentialBackoff
```

5.1.2.1.7 Request

Diese Methode wird einmalig aufgerufen, um einen Request zum Broker zu versenden.

Mögliche Fehler werden an den Ausgängen `bError` und `hrErrorCode` der Bausteininstanz ausgegeben.

Syntax

```
METHOD Request : BOOL
VAR_IN_OUT
    sTopic              : STRING; // topic string (UTF-8) with any length (attend that MQTT topics a
re case sensitive)
    sResponseTopic      :
STRING; // topic string (UTF-8) with any length (attend that MQTT topics are case sensitive)
END_VAR
VAR_INPUT
    pPayload            : PVOID;
    nPayloadSize        : UDINT;
    eQoS                : TcIotMqttQos; // quality of service between the publishing client and the
broker
    bRetain             : BOOL; // if TRUE the broker stores the message in order to send it to new
subscribers
    bQueue              : BOOL; // for future extension
    pProps              : POINTER TO MqttPublishProperties;
    pCorrelationData    : POINTER TO BYTE;
    nCorrelationDataSize: UINT;
END_VAR
```

Rückgabewert

Name	Typ	Beschreibung
Request	BOOL	Bei erfolgreichem Aufruf liefert die Methode den Rückgabewert TRUE.

Eingänge

Name	Typ	Beschreibung
pPayload	PVOID	Adresse zum Payload der MQTT-Nachricht
nPayloadSize	UDINT	Größe vom Payload in Bytes
eQoS	TcIotMqttQos	"Quality of Service"
bRetain	BOOL	Wenn bRetain TRUE ist, speichert der Broker die Nachricht, um sie später hinzukommenden Subscribern nachträglich zukommen zu lassen.
bQueue	BOOL	Reservierter Parameter, welcher immer mit FALSE belegt werden kann.
pProps	POINTER TO MqttPublishProperties	Pointer auf zu verschickende Publish Properties. Der Parameter ist optional. Um Publish Properties anzugeben kann der FB_IotMqtt5PublishProperties [▶ 110] zur Hilfe genommen werden.
pCorrelationData	POINTER TO BYTE	Pointer auf die zu verschickende CorrelationData. Hiermit kann bei einem Request/Response Verfahren eine Zuordnung von Request zu empfangener Response erfolgen.
nCorrelationDataSize	UINT	Größe der CorrelationData in Bytes

Ein-/Ausgänge

Name	Typ	Beschreibung
sTopic	STRING	Topic der MQTT-Nachricht
sResponseTopic	STRING	Response-Topic auf dem der Request beantwortet werden soll.

Message-Payload-Formatierung

Beachten Sie, dass der Datentyp und die Formatierung des Inhalts der Sender- und Empfängerseite bekannt sein müssen, insbesondere beim Versand von Binärinformationen (Alignment) oder Strings (mit/ohne Nullterminierung).

Strings im UTF-8-Format

Die hier verwendeten Variablen vom Typ STRING nutzen das UTF-8-Format. Diese STRING-Formatierung ist üblich bei IoT/MQTT-Kommunikation sowie JSON-Dokumenten.

Um Sonderzeichen und Texte verschiedenster Sprachen empfangen zu können, wird der Zeichensatz in den Bibliotheken Tc3_IotBase und Tc3_JsonXml nicht auf den typischen Zeichensatz vom Datentyp STRING beschränkt. Stattdessen wird der Unicode-Zeichensatz als UTF-8-Format in Verbindung mit dem Datentyp STRING verwendet.

Bei Verwendung des ASCII-Zeichensatzes besteht kein Unterschied zwischen der typischen Formatierung in einem STRING und der UTF-8-Formatierung eines STRING.

Weitere Informationen zum UTF-8-STRING-Format sowie vorhandenen Anzeige- und Konvertierungsmöglichkeiten finden Sie in der Dokumentation der SPS-Bibliothek Tc2_Utilities.

5.1.2.1.8 Response

Diese Methode wird einmalig aufgerufen, um eine Response zu einem Request zu versenden.

Mögliche Fehler werden an den Ausgängen `bError` und `hrErrorCode` der Bausteininstanz ausgegeben.

Syntax

```
METHOD Response : BOOL
VAR_IN_OUT
    sResponseTopic       : STRING; // topic string (UTF-8) with any length (attend that MQTT topics
are case sensitive)
END_VAR
VAR_INPUT
    pPayload             : PVOID;
    nPayloadSize         : UDINT;
    eQoS                 : TcIotMqttQos; // quality of service between the publishing client and the
broker
    bRetain              : BOOL; // if TRUE the broker stores the message in order to send it to new
subscribers
    bQueue               : BOOL; // for future extension
    pProps               : POINTER TO MqttPublishProperties;
    pCorrelationData     : POINTER TO BYTE;
    nCorrelationDataSize: UINT;
END_VAR
```

Rückgabewert

Name	Typ	Beschreibung
Response	BOOL	Bei erfolgreichem Aufruf liefert die Methode den Rückgabewert TRUE.

Eingänge

Name	Typ	Beschreibung
pPayload	PVOID	Adresse zum Payload der MQTT-Nachricht
nPayloadSize	UDINT	Größe vom Payload in Bytes
eQoS	TcIotMqttQos	"Quality of Service"
bRetain	BOOL	Wenn bRetain TRUE ist, speichert der Broker die Nachricht, um sie später hinzukommenden Subscribern nachträglich zukommen zu lassen.
bQueue	BOOL	Reservierter Parameter, welcher immer mit FALSE belegt werden kann.
pProps	POINTER TO MqttPublishProperties	Pointer auf zu verschickende Publish Properties. Der Parameter ist optional. Um Publish Properties anzugeben kann der FB_IotMqtt5PublishProperties [▶ 110] zur Hilfe genommen werden.
pCorrelationData	POINTER TO BYTE	Pointer auf die zu verschickende CorrelationData. Hiermit kann bei einem Request/Response Verfahren eine Zuordnung von Request zu empfangener Response erfolgen.
nCorrelationDataSize	UINT	Größe der CorrelationData in Bytes

Ein-/Ausgänge

Name	Typ	Beschreibung
sResponseTopic	STRING	Topic auf dem die Response gesendet werden soll.

Message-Payload-Formatierung

Beachten Sie, dass der Datentyp und die Formatierung des Inhalts der Sender- und Empfängerseite bekannt sein müssen, insbesondere beim Versand von Binärinformationen (Alignment) oder Strings (mit/ohne Nullterminierung).

Strings im UTF-8-Format

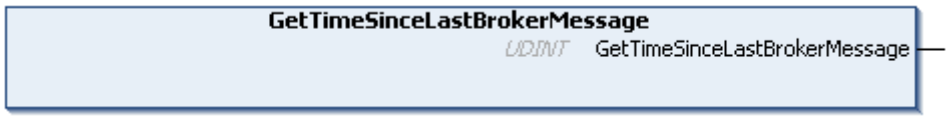
Die hier verwendeten Variablen vom Typ STRING nutzen das UTF-8-Format. Diese STRING-Formatierung ist üblich bei IoT/MQTT-Kommunikation sowie JSON-Dokumenten.

Um Sonderzeichen und Texte verschiedenster Sprachen empfangen zu können, wird der Zeichensatz in den Bibliotheken Tc3_IotBase und Tc3_JsonXml nicht auf den typischen Zeichensatz vom Datentyp STRING beschränkt. Stattdessen wird der Unicode-Zeichensatz als UTF-8-Format in Verbindung mit dem Datentyp STRING verwendet.

Bei Verwendung des ASCII-Zeichensatzes besteht kein Unterschied zwischen der typischen Formatierung in einem STRING und der UTF-8-Formatierung eines STRING.

Weitere Informationen zum UTF-8-STRING-Format sowie vorhandenen Anzeige- und Konvertierungsmöglichkeiten finden Sie in der Dokumentation der SPS-Bibliothek Tc2_Utilities.

5.1.2.1.9 GetTimeSinceLastBrokerMessage



Diese Methode gibt die Zeit (in ms) seit der letzten Nachricht vom Message Broker an. Bei jeder neuen Nachricht vom Message Broker wird die Zeit wieder auf 0 zurückgesetzt. Hierbei ist es egal, ob es sich um Ping-Nachrichten oder reguläre Publish/Subscribe-Nachrichten handelt.

In der MQTT-Spezifikation ist definiert, dass ein MQTT-Client selbst bestimmt nach welcher Zeit er einen Timeout in Richtung des Message Brokers annimmt. Das ist mit dieser Methode aus der SPS möglich. Die Keep Alive-Zeit spezifiziert hingegen nach welcher Zeit (mit 1,5 multipliziert) der Message Broker bei ausbleibenden Nachrichten vom Client einen Timeout des Clients annimmt.

Syntax

```
METHOD GetTimeSinceLastBrokerMessage : UDINT
```

Rückgabewert

Name	Typ	Beschreibung
GetTimeSinceLastBrokerMessage	UDINT	Die Zeit in Millisekunden seit der letzten Nachricht vom Message Broker.

5.1.2.2 FB_IotMqtt5ClientBase

Der Funktionsbaustein ermöglicht die Kommunikation zu einem Message Broker basierend auf MQTT Version 5.0.

Ein Client-Funktionsbaustein ist hierbei für die Verbindung zu genau einem Broker zuständig. Um die Hintergrundkommunikation zu diesem Broker zu gewährleisten und Nachrichten empfangen zu können, muss die Methode Execute() des Funktionsbausteins zyklisch aufgerufen werden.

Alle Verbindungsparameter sind als Eingangsparameter vorhanden und werden beim Verbindungsaufbau ausgewertet.

Verwenden Sie den `FB_IotMqtt5ClientBase`, wenn Sie die Callback-Methoden (`OnMqtt5Message()` und andere) selbst implementieren möchten. Verwenden Sie den FB_IotMqtt5Client, wenn Sie keine Callback-Methoden selbst implementieren möchten und eingehende neue Nachrichten in der am Ausgang vorhandenen Message Queue bereitgestellt werden sollen.

Syntax

Definition:

```
FUNCTION_BLOCK FB_IotMqtt5Client
VAR_INPUT
    {attribute 'TcEncoding':='UTF-8'}
    sClientId        : STRING(255);        // default is generated during initialization
    {attribute 'TcEncoding':='UTF-8'}
    sHostName        : STRING(255) := '127.0.0.1'; // default is local host
    nHostPort        : UINT := 1883;       // default is 1883
    {attribute 'TcEncoding':='UTF-8'}
    sTopicPrefix     : STRING(255);        // topic prefix for pub and sub of this client (handled i
nternally)
    nKeepAlive       : UINT := 60;         // in seconds

    {attribute 'TcEncoding':='UTF-8'}
    sUserName        : STRING(255);        // optional parameter
    {attribute 'TcEncoding':='UTF-8'}
    sUserPassword    : STRING(255);        // optional parameter
    stWill           : ST_IotMqtt5Will;    // optional parameter
    stTLS            : ST_IotMqtt5Tls;     // optional parameter
    stAuth           : ST_IotMqtt5Auth;    // optional parameter
    stConnect        : ST_IotMqtt5Connect; // optional parameter
END_VAR
VAR_OUTPUT
    bError           : BOOL;
    hrErrorCode      : HRESULT;
    eConnectionState : ETcIotMqttClientState;
    bConnected       : BOOL; // TRUE if connection to host is established
END_VAR
```

Eingänge

Name	Typ	Beschreibung
sClientId	STRING(255)	Die Client-ID kann individuell angegeben werden und muss bei den meisten Message Brokern eindeutig sein. Bei fehlender Angabe wird eine ID automatisch vom TwinCAT Treiber generiert, welche auf dem SPS Projektnamen basiert.
sHostName	STRING(255)	sHostName kann als Name oder als IP-Adresse angegeben werden. Bei fehlender Angabe wird Local Host verwendet.
nHostPort	UINT	Hier wird der Host Port angegeben. Dieser ist standardmäßig 1883.
sTopicPrefix	STRING(255)	Hier kann ein Topic Prefix angegeben werden, das automatisch für alle Publish- und Subscribe-Kommandos angefügt wird.
nKeepAlive	UINT	Hier kann die Watchdog-Zeit in Sekunden angegeben werden, mit der die Verbindung zwischen Client und Broker überwacht wird.
sUserName	STRING(255)	Optional kann ein Benutzername angegeben werden.
sUserPassword	STRING(255)	Zum Benutzernamen kann hier ein Passwort angegeben werden.
stWill	ST_IotMqtt5Will [▶ 94]	Bei irregulärem Verbindungsabbau des Clients vom Broker kann optional eine letzte vorkonfigurierte Nachricht vom Broker an das sogenannte „Will-Topic“ versendet werden. Hier wird diese Nachricht spezifiziert.
stTLS	ST_IotMqtt5Tls [▶ 95]	Wenn der Broker eine TLS gesicherte Verbindung anbietet, kann hier die nötige Konfiguration vorgenommen werden.
stAuth	ST_IotMqtt5Auth [▶ 96]	Mit dieser Struktur können erweiterte Authentifizierungsinformationen an den Message Broker übergeben werden.
stConnect	ST_IotMqtt5Connect [▶ 97]	Mit dieser Struktur können erweiterte Verbindungseinstellungen an den Message Broker übergeben werden.

Ausgänge

Name	Typ	Beschreibung
bError	BOOL	Wird TRUE, sobald eine Fehlersituation eintritt.
hrErrorCode	HRESULT	Liefert bei einem gesetzten bError-Ausgang einen Fehlercode. Eine Erläuterung zu den möglichen Fehlercodes befindet sich im Anhang.
eConnectionState	ETcIotMqttClientState [▶ 115]	Gibt den Zustand der Verbindung vom Client zum Broker als Enumeration ETcIotMqttClientState an.
bConnected	BOOL	Dieser Ausgang ist TRUE, wenn eine Verbindung vom Client zum Broker besteht.

Methoden

Name	Beschreibung
Execute [▶ 85]	Methode zur Hintergrundkommunikation mit dem TwinCAT-Treiber. Die Methode muss zyklisch aufgerufen werden.
Publish [▶ 85]	Methode zum Ausführen eines Publish an einen MQTT Message Broker.
Request [▶ 91]	Hiermit kann ein Request im Rahmen des Request/Response Verfahrens von MQTTv5 verschickt werden.
Response [▶ 93]	Hiermit kann eine Response im Rahmen des Request/Response Verfahrens von MQTTv5 verschickt werden.
Subscribe [▶ 86]	Methode zum Einrichten einer Subscription.
Unsubscribe [▶ 87]	Methode zum Entfernen einer Subscription.
ActivateExponentialBackoff [▶ 90]	Aktiviert die Funktion des exponential backoff [▶ 56]
DeactivateExponentialBackoff [▶ 91]	Deaktiviert die Funktion des exponential backoff [▶ 56]
GetTimeSinceLastBrokerMessage [▶ 94]	Methode zur Abfrage der Zeit (in ms) seit der letzten Nachricht vom Message Broker.

Ereignisgesteuerte Methoden (Callback-Methoden)

Name	Beschreibung
OnMqtt5Authentication	Callback-Methode, die vom TwinCAT-Treiber aufgerufen wird, wenn der Message Broker eine AUTH-Nachricht als Antwort auf einen Connect an den Client sendet.
OnMqtt5ConnAck	Callback-Methode, die vom TwinCAT-Treiber aufgerufen wird, wenn der Message Broker eine CONNACK-Nachricht als Antwort auf einen Connect verschickt.
OnMqtt5Disconnected	Callback-Methode, die vom TwinCAT-Treiber aufgerufen wird, wenn die Verbindung zu einem Message Broker getrennt wurde.
OnMqtt5Message	Callback-Methode, die vom TwinCAT-Treiber aufgerufen wird, wenn eine Subscription auf ein Topic erfolgt ist und eingehende Nachrichten empfangen werden.

Message-Payload-Formatierung

Beachten Sie, dass der Datentyp und die Formatierung des Inhalts der Sender- und Empfängerseite bekannt sein müssen, insbesondere beim Versand von Binärinformationen (Alignment) oder Strings (mit/ohne Nullterminierung).

Strings im UTF-8-Format

Die hier verwendeten Variablen vom Typ STRING nutzen das UTF-8-Format. Diese STRING-Formatierung ist üblich bei IoT/MQTT-Kommunikation sowie JSON-Dokumenten.

Um Sonderzeichen und Texte verschiedenster Sprachen empfangen zu können, wird der Zeichensatz in den Bibliotheken Tc3_IotBase und Tc3_JsonXml nicht auf den typischen Zeichensatz vom Datentyp STRING beschränkt. Stattdessen wird der Unicode-Zeichensatz als UTF-8-Format in Verbindung mit dem Datentyp STRING verwendet.

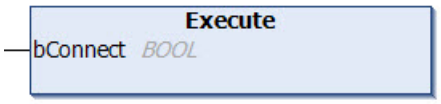
Bei Verwendung des ASCII-Zeichensatzes besteht kein Unterschied zwischen der typischen Formatierung in einem STRING und der UTF-8-Formatierung eines STRING.

Weitere Informationen zum UTF-8-STRING-Format sowie vorhandenen Anzeige- und Konvertierungsmöglichkeiten finden Sie in der Dokumentation der SPS-Bibliothek Tc2_Utilities.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4026.0	IPC oder CX (x86, x64, ARM)	Tc3_IotBase (>= v3.4.2.0)

5.1.2.2.1 Execute



Diese Methode muss zyklisch aufgerufen werden, um die Hintergrundkommunikation mit dem MQTT Broker zu gewährleisten.

Mögliche Fehler werden an den Ausgängen `bError` und `hrErrorCode` der Bausteininstanz ausgegeben.

Syntax

```
METHOD Execute
VAR_INPUT
    bConnect : BOOL;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
bConnect	BOOL	Wenn bConnect auf TRUE gesetzt wird, findet der Verbindungsaufbau zum Broker statt. Um die Verbindung zu halten, muss bConnect gesetzt bleiben.

5.1.2.2.2 Publish

Diese Methode wird einmalig aufgerufen, um eine Nachricht zum Broker zu versenden.

Mögliche Fehler werden an den Ausgängen `bError` und `hrErrorCode` der Bausteininstanz ausgegeben.

Syntax

```
METHOD Publish : BOOL
VAR_IN_OUT
    sTopic       : STRING; // topic string (UTF-8) with any length (attend that MQTT topics are case
 sensitive)
END_VAR
VAR_INPUT
    pPayload     : PVOID;
    nPayloadSize : UDINT;
    eQoS         : TcIotMqttQos; // quality of service between the publishing client and the broker
    bRetain      : BOOL; // if TRUE the broker stores the message in order to send it to new subscri
bers
    bQueue       : BOOL; // for future extension
    pProps       : POINTER TO MqttPublishProperties; // optional
END_VAR
```

Rückgabewert

Name	Typ	Beschreibung
Publish	BOOL	Bei erfolgreichem Aufruf liefert die Methode den Rückgabewert TRUE.

Eingänge

Name	Typ	Beschreibung
pPayload	PVOID	Adresse zum Payload der MQTT-Nachricht
nPayloadSize	UDINT	Größe vom Payload in Bytes
eQoS	TcIotMqttQos	"Quality of Service"
bRetain	BOOL	Wenn bRetain TRUE ist, speichert der Broker die Nachricht, um sie später hinzukommenden Subscribern nachträglich zukommen zu lassen.
bQueue	BOOL	Reservierter Parameter, welcher immer mit FALSE belegt werden kann.
pProps	POINTER TO MqttPublishProperties	Pointer auf zu verschickende Publish Properties. Der Parameter ist optional. Um Publish Properties anzugeben kann der FB_IotMqtt5PublishProperties [▶ 110] zur Hilfe genommen werden.

Ein-/Ausgänge

Name	Typ	Beschreibung
sTopic	STRING	Topic der MQTT-Nachricht

Message-Payload-Formatierung

Beachten Sie, dass der Datentyp und die Formatierung des Inhalts der Sender- und Empfängerseite bekannt sein müssen, insbesondere beim Versand von Binärinformationen (Alignment) oder Strings (mit/ohne Nullterminierung).

Strings im UTF-8-Format

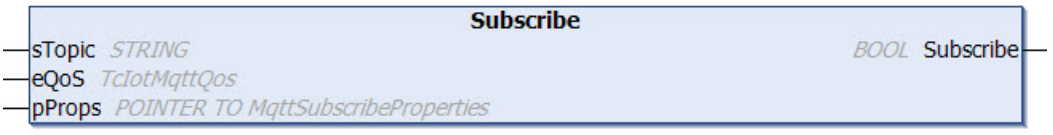
Die hier verwendeten Variablen vom Typ STRING nutzen das UTF-8-Format. Diese STRING-Formatierung ist üblich bei IoT/MQTT-Kommunikation sowie JSON-Dokumenten.

Um Sonderzeichen und Texte verschiedenster Sprachen empfangen zu können, wird der Zeichensatz in den Bibliotheken Tc3_IotBase und Tc3_JsonXml nicht auf den typischen Zeichensatz vom Datentyp STRING beschränkt. Stattdessen wird der Unicode-Zeichensatz als UTF-8-Format in Verbindung mit dem Datentyp STRING verwendet.

Bei Verwendung des ASCII-Zeichensatzes besteht kein Unterschied zwischen der typischen Formatierung in einem STRING und der UTF-8-Formatierung eines STRING.

Weitere Informationen zum UTF-8-STRING-Format sowie vorhandenen Anzeige- und Konvertierungsmöglichkeiten finden Sie in der Dokumentation der SPS-Bibliothek Tc2_Utilities.

5.1.2.2.3 Subscribe



Mit dieser Methode meldet der Client dem Broker, dass er alle MQTT-Nachrichten mit einem bestimmten Topic erhalten möchte. Die Methode kann eine MQTT-Client-Instanz auf mehrere Topics anmelden.

Mögliche Fehler werden an den Ausgängen `bError` und `hrErrorCode` der Bausteininstanz ausgegeben.

Syntax

```
METHOD Subscribe : BOOL
VAR_IN_OUT
    sTopic    : STRING; // topic string (UTF-8) with any length (attend that MQTT topics are case se
nsitive)
END_VAR
VAR_INPUT
    eQoS      : TcIotMqttQos; // quality of service between the publishing client and the broker
    pProps    : POINTER TO MqttSubscribeProperties; // optional
END_VAR
```

Rückgabewert

Name	Typ	Beschreibung
Subscribe	BOOL	Bei erfolgreichem Aufruf liefert die Methode den Rückgabewert TRUE.

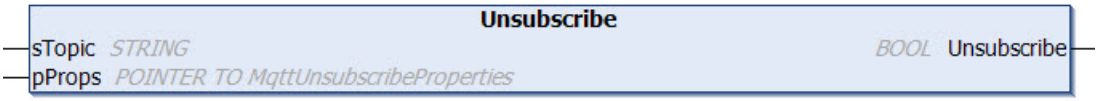
Eingänge

Name	Typ	Beschreibung
eQoS	TcIotMqttQos	"Quality of Service"
pProps	POINTER TO MqttSubscribeProperties	Pointer auf zu verschickende Subscribe Properties. Der Parameter ist optional. Um Subscribe Properties anzugeben kann der FB_IotMqtt5SubscribeProperties [▶ 112] zur Hilfe genommen werden.

Ein-/Ausgänge

Name	Typ	Beschreibung
sTopic	STRING	Topic der MQTT-Nachricht

5.1.2.2.4 Unsubscribe



Mit dieser Methode meldet der Client dem Broker, dass keine weiteren Nachrichten mit dem angegebenen Topic mehr zu ihm übertragen werden sollen.

Mögliche Fehler werden an den Ausgängen `bError` und `hrErrorCode` der Bausteininstanz ausgegeben.

Syntax

```
METHOD Unsubscribe : BOOL
VAR_IN_OUT
    sTopic    : STRING; // topic string (UTF-8) with any length (attend that MQTT topics are case se
nsitive)
END_VAR
VAR_INPUT
    pProps    : POINTER TO MqttUnsubscribeProperties; // optional
END_VAR
```

Rückgabewert

Name	Typ	Beschreibung
Unsubscribe	BOOL	Bei erfolgreichem Aufruf liefert die Methode den Rückgabewert TRUE.

Eingänge

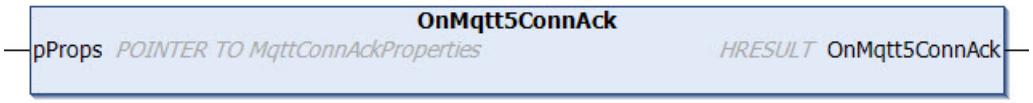
Name	Typ	Beschreibung
pProps	POINTER TO MqttUnsubscribeProperties	Pointer auf zu verschickende Unsubscribe Properties. Der Parameter ist optional. Um Unsubscribe Properties anzugeben kann der FB_IotMqtt5UnsubscribeProperties [▶ 113] zur Hilfe genommen werden.

Ein-/Ausgänge

Name	Typ	Beschreibung
sTopic	STRING	Topic, mit dem keine weiteren Nachrichten empfangen werden sollen.

5.1.2.2.5 OnMqtt5ConnAck

Callback-Methode



Diese Methode darf nicht vom Anwender aufgerufen werden. Stattdessen kann vom Funktionsbaustein FB_IotMqtt5ClientBase abgeleitet und diese Methode überschrieben werden. Während dem Aufruf der Methode Execute() hat der zuständige TwinCAT-Treiber die Möglichkeit, über diesen Callback eine Benachrichtigung zu empfangen, wenn der Message Broker ein CONNACK als Antwort auf einen Connect verschickt hat.

Syntax

```
METHOD OnMqtt5Disconnected : HRESULT
VAR_INPUT
    pProps : POINTER TO MqttConnAckProperties;
END_VAR
```

Rückgabewert

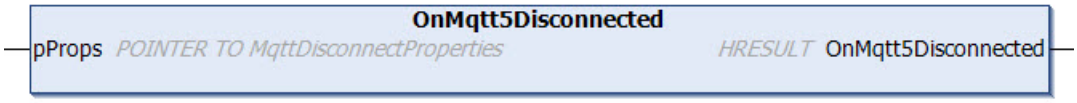
Name	Typ	Beschreibung
OnMqtt5ConnAck	HRESULT	

Eingänge

Name	Typ	Beschreibung
pProps	POINTER TO MqttConnAck Properties	MQTT Properties, die beim CONNACK empfangen wurden.

5.1.2.2.6 OnMqtt5Disconnected

Callback-Methode



Diese Methode darf nicht vom Anwender aufgerufen werden. Stattdessen kann vom Funktionsbaustein FB_IotMqtt5ClientBase abgeleitet und diese Methode überschrieben werden. Während dem Aufruf der Methode Execute()hat der zuständige TwinCAT-Treiber die Möglichkeit, im Fall eines Verbindungsabbruchs mit dem Message Broker, die Methode OnMqtt5Disconnected() aufzurufen.

Syntax

```
METHOD OnMqtt5Disconnected : HRESULT
VAR_INPUT
    pProps : POINTER TO MqttDisconnectProperties;
END_VAR
```

Rückgabewert

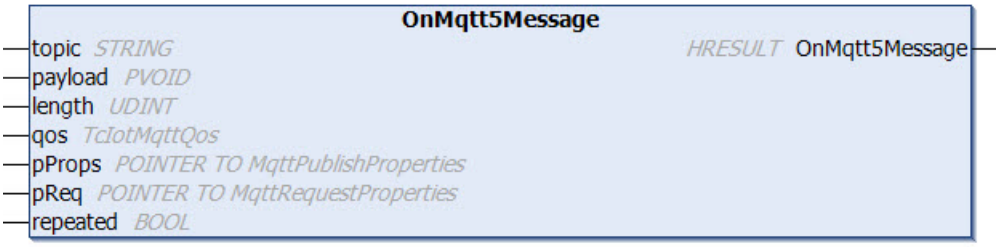
Name	Typ	Beschreibung
OnMqtt5Disconnect	HRESULT	

Eingänge

Name	Typ	Beschreibung
pProps	POINTER TO MqttDisconnectProperties	MQTT Properties, die beim Disconnect empfangen wurden.

5.1.2.2.7 OnMqtt5Message

Callback-Methode



Diese Methode darf nicht vom Anwender aufgerufen werden. Stattdessen kann vom Funktionsbaustein FB_IotMqtt5ClientBase abgeleitet und diese Methode überschrieben werden. Während dem Aufruf der Methode Execute()hat der zuständige TwinCAT-Treiber die Möglichkeit, im Fall von neuen eingehenden Nachrichten, die Methode OnMqtt5Message() aufzurufen. Bei mehreren eingehenden Nachrichten wird die Callback-Methode mehrfach, je Nachricht einmal, aufgerufen. Die Implementierung der Methode muss dies berücksichtigen.
Die Verwendung der Callback-Methode ist auch im Beispiel IotMqttSampleUsingCallback [▶ 321] erläutert.

Syntax

```
METHOD OnMqtt5Message : HRESULT
VAR_IN_OUT CONSTANT
    topic    : STRING;
END_VAR
VAR_INPUT
    payload  : PVOID;
    length   : UDINT;
    qos      : TcIotMqttQos;
    pProps   : POINTER TO MqttPublishProperties;
    pReq     : POINTER TO MqttRequestProperties;
    repeated : BOOL;
END_VAR
```

Rückgabewert

Name	Typ	Beschreibung
OnMqtt5Message	HRESULT	Der Rückgabewert der Methode ist mit S_OK zu belegen, sofern die Nachricht angenommen wurde. Soll die Nachricht im Kontext des nächsten Execute()-Aufrufs erneut ausgegeben werden, kann der Rückgabewert mit S_FALSE belegt werden.

Eingänge

Name	Typ	Beschreibung
payload	PVOID	Adresse des Payload der eingegangenen MQTT-Nachricht
length	UDINT	Größe vom Payload in Bytes
qos	TcIotMqttQos	"Quality of Service"
pProps	POINTER TO MqttPublishProperties	MQTT Properties, die ggf. mit einer Nachricht empfangen wurden.
pReq	POINTER TO MqttRequestProperties	MQTT Request Properties, der ggf. mit einer Nachricht empfangen wurde.
repeated	BOOL	Wurde der letzte OnMqtt5Message()-Methodenaufruf vom Anwender nicht mit S_OK erwidert, wird die Nachricht im Kontext des nächsten Execute()-Aufrufs erneut ausgegeben und hierbei der Parameter repeated gesetzt. Daran ist zu erkennen, dass die Nachricht wiederholt ausgegeben wird.

Ein-/Ausgänge

Name	Typ	Beschreibung
topic	STRING	Topic der eingegangenen MQTT-Nachricht

5.1.2.2.8 ActivateExponentialBackoff

Um im Falle eines Verbindungsfehlers mit dem Message Broker diesen nicht unnötig mit Verbindungsanforderungen zu belasten, kann auf eine Funktionalität namens "exponential backoff" zurückgegriffen werden. Hierbei wird nach einem TLS-Verbindungsfehler mit dem Message Broker die Reconnect-Rate multiplikativ angepasst. Diese Funktion ist über die Methode ActivateExponentialBackoff() [▶ 63] aktivierbar. Die Parameter der Methode geben hierbei die Minimal- und Maximalzeit für den Algorithmus an. Die Minimalzeit beschreibt den Anfangsverzögerungswert für den erneuten Verbindungsversuch. Die Maximalzeit beschreibt den größten Verzögerungswert. Die Verzögerungswerte werden bis zum Erreichen des Maximalwerts verdoppelt. Sobald eine Verbindung hergestellt werden konnte, wird die Backoff-Rate wieder auf den Ursprungswert zurückgesetzt. Über die Methode DeactivateExponentialBackoff() [▶ 64] kann diese Funktion auch programmatisch wieder außer Kraft gesetzt werden.

Syntax

```
METHOD ActivateExponentialBackoff
VAR_INPUT
    tMqttBackoffMinTime: TIME;
    tMqttBackoffMaxTime: TIME;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
tMqttBackoffMinTime	TIME	Beschreibt den Anfangsverzögerungswert für den erneuten Verbindungsversuch.
tMqttBackoffMaxTime	TIME	Beschreibt den größten Verzögerungswert. Nach Erreichen dieses Werts, erfolgen alle erneuten Verbindungsversuche immer in diesen Abständen.

5.1.2.2.9 DeactivateExponentialBackoff

Um im Falle eines Verbindungsfehlers mit dem Message Broker diesen nicht unnötig mit Verbindungsanforderungen zu belasten, kann auf eine Funktionalität namens "exponential backoff" zurückgegriffen werden. Hierbei wird nach einem TLS-Verbindungsfehler mit dem Message Broker die Reconnect-Rate multiplikativ angepasst. Diese Funktion ist über die Methode ActivateExponentialBackoff() [▶ 63] aktivierbar. Die Parameter der Methode geben hierbei die Minimal- und Maximalzeit für den Algorithmus an. Die Minimalzeit beschreibt den Anfangsverzögerungswert für den erneuten Verbindungsversuch. Die Maximalzeit beschreibt den größten Verzögerungswert. Die Verzögerungswerte werden bis zum Erreichen des Maximalwerts verdoppelt. Sobald eine Verbindung hergestellt werden konnte, wird die Backoff-Rate wieder auf den Ursprungswert zurückgesetzt. Über die Methode DeactivateExponentialBackoff() [▶ 64] kann diese Funktion auch programmatisch wieder außer Kraft gesetzt werden.

Syntax

```
METHOD DeactivateExponentialBackoff
```

5.1.2.2.10 Request

Diese Methode wird einmalig aufgerufen, um einen Request zum Broker zu versenden.

Mögliche Fehler werden an den Ausgängen `bError` und `hrErrorCode` der Bausteininstanz ausgegeben.

Syntax

```
METHOD Request : BOOL
VAR_IN_OUT
    sTopic              : STRING; // topic string (UTF-8) with any length (attend that MQTT topics a
re case sensitive)
    sResponseTopic      :
STRING; // topic string (UTF-8) with any length (attend that MQTT topics are case sensitive)
END_VAR
VAR_INPUT
    pPayload            : PVOID;
    nPayloadSize        : UDINT;
    eQoS                : TcIotMqttQos; // quality of service between the publishing client and the
broker
    bRetain             : BOOL; // if TRUE the broker stores the message in order to send it to new
subscribers
    bQueue              : BOOL; // for future extension
    pProps              : POINTER TO MqttPublishProperties;
    pCorrelationData    : POINTER TO BYTE;
    nCorrelationDataSize: UINT;
END_VAR
```

Rückgabewert

Name	Typ	Beschreibung
Request	BOOL	Bei erfolgreichem Aufruf liefert die Methode den Rückgabewert TRUE.

Eingänge

Name	Typ	Beschreibung
pPayload	PVOID	Adresse zum Payload der MQTT-Nachricht
nPayloadSize	UDINT	Größe vom Payload in Bytes
eQoS	TcIotMqttQos	"Quality of Service"
bRetain	BOOL	Wenn bRetain TRUE ist, speichert der Broker die Nachricht, um sie später hinzukommenden Subscribern nachträglich zukommen zu lassen.
bQueue	BOOL	Reservierter Parameter, welcher immer mit FALSE belegt werden kann.
pProps	POINTER TO MqttPublishProperties	Pointer auf zu verschickende Publish Properties. Der Parameter ist optional. Um Publish Properties anzugeben kann der FB_IotMqtt5PublishProperties [▶ 110] zur Hilfe genommen werden.
pCorrelationData	POINTER TO BYTE	Pointer auf die zu verschickende CorrelationData. Hiermit kann bei einem Request/Response Verfahren eine Zuordnung von Request zu empfangener Response erfolgen.
nCorrelationDataSize	UINT	Größe der CorrelationData in Bytes

Ein-/Ausgänge

Name	Typ	Beschreibung
sTopic	STRING	Topic der MQTT-Nachricht
sResponseTopic	STRING	Response-Topic auf dem der Request beantwortet werden soll.

Message-Payload-Formatierung

Beachten Sie, dass der Datentyp und die Formatierung des Inhalts der Sender- und Empfängerseite bekannt sein müssen, insbesondere beim Versand von Binärinformationen (Alignment) oder Strings (mit/ohne Nullterminierung).

Strings im UTF-8-Format

Die hier verwendeten Variablen vom Typ STRING nutzen das UTF-8-Format. Diese STRING-Formatierung ist üblich bei IoT/MQTT-Kommunikation sowie JSON-Dokumenten.

Um Sonderzeichen und Texte verschiedenster Sprachen empfangen zu können, wird der Zeichensatz in den Bibliotheken Tc3_IotBase und Tc3_JsonXml nicht auf den typischen Zeichensatz vom Datentyp STRING beschränkt. Stattdessen wird der Unicode-Zeichensatz als UTF-8-Format in Verbindung mit dem Datentyp STRING verwendet.

Bei Verwendung des ASCII-Zeichensatzes besteht kein Unterschied zwischen der typischen Formatierung in einem STRING und der UTF-8-Formatierung eines STRING.

Weitere Informationen zum UTF-8-STRING-Format sowie vorhandenen Anzeige- und Konvertierungsmöglichkeiten finden Sie in der Dokumentation der SPS-Bibliothek Tc2_Utilities.

5.1.2.2.11 Response

Diese Methode wird einmalig aufgerufen, um eine Response zu einem Request zu versenden.

Mögliche Fehler werden an den Ausgängen `bError` und `hrErrorCode` der Bausteininstanz ausgegeben.

Syntax

```
METHOD Response : BOOL
VAR_IN_OUT
    sResponseTopic       : STRING; // topic string (UTF-8) with any length (attend that MQTT topics
are case sensitive)
END_VAR
VAR_INPUT
    pPayload             : PVOID;
    nPayloadSize         : UDINT;
    eQoS                 : TcIotMqttQos; // quality of service between the publishing client and the
broker
    bRetain              : BOOL; // if TRUE the broker stores the message in order to send it to new
subscribers
    bQueue               : BOOL; // for future extension
    pProps               : POINTER TO MqttPublishProperties;
    pCorrelationData     : POINTER TO BYTE;
    nCorrelationDataSize: UINT;
END_VAR
```

Rückgabewert

Name	Typ	Beschreibung
Response	BOOL	Bei erfolgreichem Aufruf liefert die Methode den Rückgabewert TRUE.

Eingänge

Name	Typ	Beschreibung
pPayload	PVOID	Adresse zum Payload der MQTT-Nachricht
nPayloadSize	UDINT	Größe vom Payload in Bytes
eQoS	TcIotMqttQos	"Quality of Service"
bRetain	BOOL	Wenn bRetain TRUE ist, speichert der Broker die Nachricht, um sie später hinzukommenden Subscribern nachträglich zukommen zu lassen.
bQueue	BOOL	Reservierter Parameter, welcher immer mit FALSE belegt werden kann.
pProps	POINTER TO MqttPublishProperties	Pointer auf zu verschickende Publish Properties. Der Parameter ist optional. Um Publish Properties anzugeben kann der FB_IotMqtt5PublishProperties [▶ 110] zur Hilfe genommen werden.
pCorrelationData	POINTER TO BYTE	Pointer auf die zu verschickende CorrelationData. Hiermit kann bei einem Request/Response Verfahren eine Zuordnung von Request zu empfangener Response erfolgen.
nCorrelationDataSize	UINT	Größe der CorrelationData in Bytes

Ein-/Ausgänge

Name	Typ	Beschreibung
sResponseTopic	STRING	Topic auf dem die Response gesendet werden soll.

Message-Payload-Formatierung

Beachten Sie, dass der Datentyp und die Formatierung des Inhalts der Sender- und Empfängerseite bekannt sein müssen, insbesondere beim Versand von Binärinformationen (Alignment) oder Strings (mit/ohne Nullterminierung).

Strings im UTF-8-Format

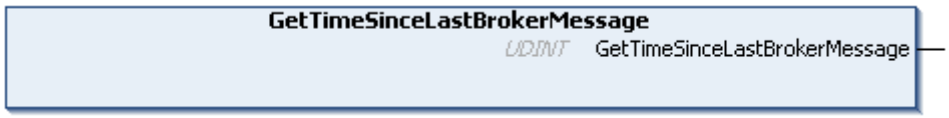
Die hier verwendeten Variablen vom Typ STRING nutzen das UTF-8-Format. Diese STRING-Formatierung ist üblich bei IoT/MQTT-Kommunikation sowie JSON-Dokumenten.

Um Sonderzeichen und Texte verschiedenster Sprachen empfangen zu können, wird der Zeichensatz in den Bibliotheken Tc3_IotBase und Tc3_JsonXml nicht auf den typischen Zeichensatz vom Datentyp STRING beschränkt. Stattdessen wird der Unicode-Zeichensatz als UTF-8-Format in Verbindung mit dem Datentyp STRING verwendet.

Bei Verwendung des ASCII-Zeichensatzes besteht kein Unterschied zwischen der typischen Formatierung in einem STRING und der UTF-8-Formatierung eines STRING.

Weitere Informationen zum UTF-8-STRING-Format sowie vorhandenen Anzeige- und Konvertierungsmöglichkeiten finden Sie in der Dokumentation der SPS-Bibliothek Tc2_Utilities.

5.1.2.2.12 GetTimeSinceLastBrokerMessage



Diese Methode gibt die Zeit (in ms) seit der letzten Nachricht vom Message Broker an. Bei jeder neuen Nachricht vom Message Broker wird die Zeit wieder auf 0 zurückgesetzt. Hierbei ist es egal, ob es sich um Ping-Nachrichten oder reguläre Publish/Subscribe-Nachrichten handelt.

In der MQTT-Spezifikation ist definiert, dass ein MQTT-Client selbst bestimmt nach welcher Zeit er einen Timeout in Richtung des Message Brokers annimmt. Das ist mit dieser Methode aus der SPS möglich. Die Keep Alive-Zeit spezifiziert hingegen nach welcher Zeit (mit 1,5 multipliziert) der Message Broker bei ausbleibenden Nachrichten vom Client einen Timeout des Clients annimmt.

Syntax

```
METHOD GetTimeSinceLastBrokerMessage : UDINT
```

Rückgabewert

Name	Typ	Beschreibung
GetTimeSinceLastBrokerMessage	UDINT	Die Zeit in Millisekunden seit der letzten Nachricht vom Message Broker.

5.1.2.3 ST_IotMqtt5Will

Mittels folgender Angaben kann eine Nachricht spezifiziert werden, die bei einem irregulärem Verbindungsabbau zwischen Client und Broker als letzte Nachricht vom Broker zu den Subscribern versendet wird, die das entsprechende Topic abonniert haben. Diese Struktur ist eine Erweiterung der bisherigen Struktur ST_IotMqttWill [▶ 65] für MQTT V5.

Keine Instanziierung und Zuweisung dieser Struktur

Diese Struktur erlaubt keine Instanziierung und Zuweisung zum FB_IotMqtt5Client bzw. FB_IotMqtt5ClientBase. Stattdessen wird der Eingangsparameter des MQTT v5 Client Funktionsbausteines direkt verwendet.

Syntax

Definition:

```
TYPE ST_IotMqtt5Will :
STRUCT
    {attribute 'TcEncoding':='UTF-8'}
    sTopic             : STRING(255);
    fbPayload          : FB_IotDataBuffer;
    eQoS               : TcIotMqttQos := TcIotMqttQos.ExactlyOnceDelivery;
    bRetain            : BOOL;
    {attribute 'TcEncoding':='UTF-8'}
    sContentType       : STRING(255);
    {attribute 'TcEncoding':='UTF-8'}
    sResponseTopic     : STRING(255);
    nMsgExpiryInterval : UDINT;
    nDelay             : UDINT;
    bPayloadUtf8       : BOOL;
    fbCorrelationData  : FB_IotDataBuffer;
    fbUserProperties   : FB_IotMqtt5UserProperties;
END_STRUCT
END_TYPE
```

Parameter

Name	Typ	Beschreibung
sTopic	STRING(255)	Topic der Nachricht.
fbPayload	FB_IotDataBuffer	Datenpuffer für den Payload der Nachricht.
eQoS	TcIotMqttQos	Die „Quality of Service“ bietet folgende Einstellmöglichkeiten: QoS-Level 0, QoS-Level 1, QoS-Level 2 (Siehe QoS).
bRetain	BOOL	Wenn bRetain TRUE ist, speichert der Broker die Nachricht, um sie später hinzukommenden Subscribern nachträglich zukommen zu lassen.
sContentType	STRING(255)	Beschreibt das Encoding des Payloads. Hier eignen sich für hohe Kompatibilität vor allem MIME-Typen wie bspw. „text/plain“.
sResponseTopic	STRING(255)	An dieser Stelle kann das Antwort-Topic angegeben werden, auf dem eine Rückantwort eines anderen Clients erwartet wird.
nMsgExpiryInterval	UDINT	Zeit in Sekunden, nach der eine Last Will-Nachricht abläuft und bei einem Verbindungsabbruch nicht mehr durch den Broker zugestellt wird.
nDelay	UDINT	Verzögerungsintervall in Sekunden bis die LastWill Nachricht verschickt wird.
bPayloadUtf8	BOOL	Formatindikator für den Nachrichteninhalt.
fbCorrelationData	FB_IotDataBuffer	Datenpuffer für CorrelationData
fbUserProperties	FB_IotMqtt5UserProperties [▶ 114]	An dieser Stelle können UserProperties angegeben werden.

5.1.2.4 ST_IotMqtt5Tls

TLS-Sicherheitseinstellung für den MQTT Client (V5).

Es kann entweder CA (certificate authority) oder PSK verwendet werden.

Syntax

Definition:

```
TYPE ST_IotMqtt5Tls :
STRUCT
    sCA                : STRING(255); // certificate authority as filename (PEM or DER format) or as
 string (PEM)
    sCert              : STRING(255); // client certificate as filename (PEM or DER format) or as st
ring (PEM)
    sKeyFile           : STRING(255);
    sKeyPwd            : STRING(255);
    sCrl               : STRING(255); // Certificate Revocation List as filename (PEM or DER format)
 or as string (PEM)
    sCiphers           : STRING(255);
    sVersion           : STRING(80) := 'tlsv1.2'; // TLS version
    bNoServerCertCheck : BOOL := FALSE;

    sPskIdentity       : STRING(255);
    aPskKey            : ARRAY[1..64] OF BYTE;
    nPskKeyLen         : USINT;

    sAzureSas          : STRING(511);
END_STRUCT
END_TYPE
```

Parameter

Name	Typ	Beschreibung
sCA	STRING(255)	Zertifikat der Certificate Authority (CA)
sCert	STRING(255)	Client-Zertifikat, welches zur Authentifizierung am Broker verwendet werden soll
sKeyFile	STRING(255)	Privater Schlüssel des Clients
sKeyPwd	STRING(255)	Passwort des privaten Schlüssels, falls anwendbar
sCrl	STRING(255)	Pfad zur Certificate Revocation List, welche im Format PEM oder DER vorliegen kann
sCiphers	STRING(255)	Zu verwendende Cipher Suites, angegeben im OpenSSL-String-Format
sVersion	STRING(80)	Zu verwendende TLS-Version
bNoServerCertCheck	BOOL	Deaktiviert die Überprüfung des Server-Zertifikats auf Gültigkeit. Wenn ohne TLS-Verschlüsselung kommuniziert werden soll, muss dieser Wert auf FALSE bleiben.
sPskIdentity	STRING(255)	PreSharedKey-Identität für TLS-PSK Verbindung
aPskKey	ARRAY[1..64] OF BYTE	PreSharedKey für TLS-PSK-Verbindung
nPskKeyLen	USINT	Länge des PreSharedKey in Bytes
sAzureSAS	STRING(511)	SAS Token für Verbindung mit dem Microsoft Azure IoT Hub

5.1.2.5 ST_IotMqtt5Auth

Authentifizierungseinstellungen für den MQTT Client (V5).

Syntax

Definition:

```
TYPE ST_IotMqtt5Auth :
STRUCT
    {attribute 'TcEncoding':='UTF-8'}
    sAuthMethod        : STRING(255);
    aAuthData          : ARRAY[0..4095] OF BYTE;
    nAuthDataSize      : UINT;
END_STRUCT
END_TYPE
```

Parameter

Name	Typ	Beschreibung
sAuthMethod	STRING(255)	UTF-8-codierte Bezeichnung einer erweiterten Authentifizierungsmethode.
aAuthData	ARRAY[0..4095] OF BYTE	Dieses Array enthält binäre Authentifizierungsdaten.
nAuthDataSize	UINT	Spezifiziert die Lange der Authentifizierungsdaten.

5.1.2.6 ST_IotMqtt5Connect

Verbindungseinstellungen für den MQTTv5 Client, welche am Baustein FB_IotMqtt5Client [▶ 81] gesetzt werden können. Über diese Einstellungen kann der Client dem Message Broker Informationen zu potenziellen Einschränkungen mitteilen, zum Beispiel eine maximale Paketgröße die der Client empfangen kann.

Syntax

Definition:

```
TYPE ST_IotMqtt5Connect :
STRUCT
    nSessionExpire     : UDINT;
    nMaxPacketSize     : UDINT;
    nReceiveMax        : UINT;
    nTopicAliasMax     : UINT;
    bReqResponseInfo   : BOOL;
    bIgnoreProblemInfo : BOOL;
END_STRUCT
END_TYPE
```

Parameter

Name	Typ	Beschreibung
nSessionExpire	UDINT	Spezifiziert die Zeit in Sekunden, nach der ein Message Broker eine Client-Session löscht, wenn der Client nicht verbunden ist.
nMaxPacketSize	UDINT	Mit dieser Einstellung teilt der Client dem Message Broker mit, bis zu welcher Größe in Bytes er Nachrichten empfangen will bzw. kann. Wenn eine größere Nachricht empfangen wird, beendet der Client die Verbindung.
nReceiveMax	UINT	Diese Einstellung legt fest, wie viele QoS 1 oder QoS 2-Nachrichten gleichzeitig an den Client kommuniziert werden dürfen. Gleichzeitig bedeutet in dem Fall, dass der Handshake (zwei- oder vierstufig) zur gleichen Zeit passiert.
nTopicAliasMax	UINT	Spezifiziert den höchsten Wert, den ein Topic Alias haben darf, um die Anzahl an gleichzeitig vorhandenen Topic Aliasen zu beschränken. Wenn der Wert 0 ist, heißt das, dass der Client keine Topic Aliase akzeptiert.
bReqResponseInfo	BOOL	Wenn der Wert TRUE ist, dann darf der Message Broker im CONNACK Response Information mitsenden. Wenn der Wert FALSE ist, sendet der Message Broker keine Response Information.
bIgnoreProblemInfo	BOOL	Wenn der Wert TRUE ist, dann darf der Message Broker bei jedem Paket einen Reason String oder User Properties mitsenden. Wenn der Wert FALSE ist, darf der Message Broker

5.1.2.7 FB_IotMqtt5MessageQueue

Dieser Funktionsbaustein bietet eine Message Queue für MQTT-Nachrichten, welche mit dem Baustein FB_IotMqtt5Client [▶ 71] empfangen werden. Hierzu wird eine Instanz bereits am Ausgang von `FB_IotMqtt5Client` bereitgestellt. Der Funktionsbaustein arbeitet nach dem First-in-First-out-Prinzip (FiFo). Es ist möglich, dass mehrere MQTT-Nachrichten innerhalb eines SPS Zyklus empfangen und an der Message Queue bereit gestellt werden.

Im Programmablauf kann mit dem Property `nQueuedMessages` geprüft werden, ob und wie viele Nachrichten in der Message Queue eingesammelt wurden. Diese Nachrichten werden mit der Methode `Dequeue()` aus dem FiFo entnommen. Hierbei wird demnach die älteste Nachricht zuerst ausgegeben.

Größe der MQTT Message Queue

Die Menge der maximal möglichen Nachrichten in der Queue lässt sich über den Paramater `cMaxEntriesInMqttMessageQueue` in der Parameterliste [▶ 117] der Bibliothek Tc3_IotBase einstellen. Standardmäßig sind dies 1000 Nachrichten. Weil eine zeitnahe Verarbeitung der Nachrichten in den allermeisten Fällen verlangt wird, ist eine Anpassung meist nicht notwendig.

Die MQTT Message Queue allokiert für neue Nachrichten entsprechend der Topic- und Payload-Größe neuen Speicher. Standardmäßig ist die maximale Größe einer Nachricht auf 100 KB und die Größe einer MQTT Message Queue auf 1000 KB begrenzt. Dies kann für Sonderfälle ebenso in der Parameterliste [▶ 117] angepasst werden.

Eigenschaften

Name	Typ	Zugriff	Beschreibung
bOverwriteOldestEntry	BOOL	Get, Set	Hier kann parametriert werden, ob, wenn die Warteschlange voll ist, eine neu empfangene Nachricht die älteste Nachricht überschreiben soll. Wenn ja (TRUE), geht die älteste Nachricht verloren. Wenn nein (FALSE), wird mit dem Empfang der neuen Nachricht gewartet, bis die Warteschlange Platz bietet. Eine volle Warteschlange ist ein unwahrscheinlicher Fall, wenn der Anwender für ein zügiges Auslesen der Queue mittels `Dequeue()` Methode sorgt.
nLostMessages	UDINT	Get	Gibt die Anzahl der MQTT-Nachrichten aus, welche vollständig verworfen werden mussten. Verworfene MQTT-Nachrichten können als Ursache eine volle Queue haben oder auch eine zu große Gesamtgröße einer neu empfangenen Nachricht.
nMaxSizeOfMessage	UDINT	Get	Maximale Größe in Bytes aller empfangenen MQTT-Nachrichten, welche vom FB_IotMqtt5Client empfangen und in die FB_IotMqtt5MessageQueue abgelegt werden sollten. Wird die mit dem Bibliotheksparameter [▶ 117] `ParameterList.cMaxSizeOfMqttMessage` festgelegte Maximalgröße überschritten, so versucht die Message Queue zuerst die Nachricht ohne User Properties abzuspeichern. Sollte diese Größenreduktion nicht ausreichen, muss die MQTT-Nachricht vollständig verworfen werden.
nQueuedMessages	UDINT	Get	Gibt die Anzahl der aktuell in der Queue gesammelten MQTT-Nachrichten aus. Die maximal mögliche Anzahl ist mit dem Bibliotheksparameter [▶ 117] `ParameterList.cMaxEntriesInMqttMessageQueue` festgelegt.

Methoden

Name	Beschreibung
Dequeue() [▶ 100]	Entnimmt der Warteschlange eine MQTT-Nachricht.
ResetQueue() [▶ 100]	Löscht alle Nachrichten aus der Warteschlange.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4026.0	IPC oder CX (x86, x64, ARM)	Tc3_IotBase (>= v3.4.2.0)

5.1.2.7.1 Dequeue

Wenn die Entnahme einer MQTT-Nachricht aus der Queue erfolgreich war, gibt die Methode den Rückgabewert TRUE aus. Die Menge der aktuell in der Queue gesammelten MQTT-Nachrichten (Property `nQueuedMessages`) wird durch Entnahme einer Nachricht um eins reduziert.

Syntax

```
METHOD Dequeue : BOOL
VAR_INPUT
    fbMessage : REFERENCE TO FB_IotMqtt5Message;
END_VAR
```

Rückgabewert

Name	Typ	Beschreibung
Dequeue	BOOL	Bei erfolgreichem Aufruf liefert die Methode den Rückgabewert TRUE.

Eingänge

Name	Typ	Beschreibung
fbMessage	REFERENCE TO FB_IotMqtt5Message	Für die Nachricht selbst wird die Referenz auf eine Instanz vom Typ FB_IotMqtt5Message übergeben.

5.1.2.7.2 ResetQueue

Mit Aufruf der Methode werden alle gesammelten MQTT-Nachrichten aus der Queue gelöscht.

Syntax

```
METHOD ResetQueue : BOOL
```

Rückgabewert

Name	Typ	Beschreibung
ResetQueue	BOOL	Bei erfolgreichem Aufruf liefert die Methode den Rückgabewert TRUE.

5.1.2.8 FB_IotMqtt5Message

Wenn der TwintCAT MQTT v5 Client (FB_IotMqtt5Client [▶ 71]) in Kombination mit einer Message Queue (FB_IotMqtt5MessageQueue [▶ 98]) verwendet wird, wird eine MQTT-Nachricht vom Baustein FB_IotMqtt5Message repräsentiert. Ankommende Nachrichten werden von der Message Queue gesammelt und dem Anwender in Form einer solchen Bausteininstanz zur Verfügung gestellt.

Die Topic- und die Payload-Größe sowie der Parameter „Quality of Service“ der MQTT-Nachricht sind sofort am Bausteinausgang als Properties verfügbar. Das Topic selbst sowie auch das Payload kann mittels der bereitgestellten Methoden CompareTopic(), GetTopic() und GetPayload() einfach ausgewertet oder kopiert werden.

Message-Payload-Formatierung

Beachten Sie, dass der Datentyp und die Formatierung des Inhalts der Sender- und Empfängerseite bekannt sein müssen, insbesondere beim Versand von Binärinformationen (Alignment) oder Strings (mit/ohne Nullterminierung).

Größe einer MQTT-Nachricht

Die maximale Größe einer in der SPS zu empfangenden MQTT-Nachricht ist von der Hardware-Plattform abhängig und sollte selbst auf leistungsstärkeren/größeren Plattformen einige wenige MB nicht überschreiten.

Die maximal mögliche Größe einer Nachricht lässt sich über den Parameter `cMaxSizeOfMqttMessage` in der Parameterliste [▶ 117] der Bibliothek Tc3_IotBase einstellen. Standardmäßig ist die Größe einer Nachricht auf 100 KB begrenzt.

Bei Verwendung der MQTT Message Queue allokiert diese für neue Nachrichten dynamisch neuen Speicher aus dem TwinCAT Router Speicher.

Eigenschaften

Name	Typ	Zugriff	Beschreibung
bPayloadUtf8	BOOL	Get	Falls TRUE, so handelt es sich beim Payload um einen in UTF-8 formatierten Text.
bTopicAlias	BOOL	Get	Gibt an, ob es sich bei dem Empfangs-Topic um einen Alias handelt.
eQoS	TcIotMqttQos	Get	„Quality of Service“ der MQTT-Nachricht
nCorrelationDataSize	UINT	Get	Größe der Correlation Data in Bytes
nMsgExpiryInterval	UDINT	Get	Gibt das Ablaufinterval der Nachricht in Sekunden an.
nPayloadSize	UDINT	Get	Größe vom Payload in Bytes
nSubIdCnt	UINT	Get	Gibt an wie viele Subscription Identifier empfangen wurden.
nUserPropertyCnt	UINT	Get	Anzahl von vorhandenen User Properties. Diese können mittels GetUserPropertyByIdx() ausgelesen werden.
nUserPropertyCntLost	UINT	Get	Anzahl von beim Empfang verworfenen User Properties. Ursache kann ein Überschreiten der festgelegten Maximalanzahl (siehe Bibliotheksparameter [▶ 117] `ParameterList.cMaxMqtt5UserProps`) oder eine zu große empfangene MQTT-Nachricht (siehe Bibliotheksparameter [▶ 117] `ParameterList.cMaxSizeOfMqttMessage`) sein.
nTopicSize	UINT	Get	Größe vom Topic in Bytes
sContentType	STRING	Get	Content Type der MQTT-Nachricht Sollte der Originaltext zu groß sein für dieses Property, so kann die Methode GetContentType() verwendet werden, um dennoch den vollständigen Text abzufragen.

Methoden

Name	Beschreibung
CompareTopic() [▶ 103]	Überprüft ein angegebenes Topic mit dem Topic in der MQTT-Nachricht.
GetContentType()	Liefert den Content Type der MQTT-Nachricht.
GetCorrelationData()	Liefert die Correlation Data der MQTT-Nachricht.
GetResponseTopic()	Liefert das Response Topic.
GetPayload() [▶ 104]	Liefert den Inhalt einer MQTT-Nachricht.
GetSubIds()	Liefert die zur MQTT-Nachricht empfangenen Subscription Identifier.
GetTopic() [▶ 104]	Liefert das Topic einer MQTT-Nachricht.
GetUserPropertyByIdx()	Gibt den Namen und Wert eines UserProperty zurück, welches durch dessen Position (Index) in der Liste angegeben wird.
GetUserPropertyValueByName()	Gibt den Wert eines UserProperty zurück, welches durch dessen Namen angegeben wird.

Strings im UTF-8-Format

Die hier verwendeten Variablen vom Typ STRING nutzen das UTF-8-Format. Diese STRING-Formatierung ist üblich bei IoT/MQTT-Kommunikation sowie JSON-Dokumenten.

Um Sonderzeichen und Texte verschiedenster Sprachen empfangen zu können, wird der Zeichensatz in den Bibliotheken Tc3_IotBase und Tc3_JsonXml nicht auf den typischen Zeichensatz vom Datentyp STRING beschränkt. Stattdessen wird der Unicode-Zeichensatz als UTF-8-Format in Verbindung mit dem Datentyp STRING verwendet.

Bei Verwendung des ASCII-Zeichensatzes besteht kein Unterschied zwischen der typischen Formatierung in einem STRING und der UTF-8-Formatierung eines STRING.

Weitere Informationen zum UTF-8-STRING-Format sowie vorhandenen Anzeige- und Konvertierungsmöglichkeiten finden Sie in der Dokumentation der SPS-Bibliothek Tc2_Utilities.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4026.0	IPC oder CX (x86, x64, ARM)	Tc3_IotBase (>= v3.4.2.0)

5.1.2.8.1 CompareTopic

Die Methode liefert TRUE als Rückgabewert, wenn das angegebene Topic mit dem Topic der MQTT-Nachricht im Funktionsbaustein identisch ist.

Syntax

```
METHOD CompareTopic : BOOL
VAR_IN_OUT CONSTANT
    sTopic : STRING; // topic string with any length (attend that MQTT topics are case sensitive)
END_VAR
```

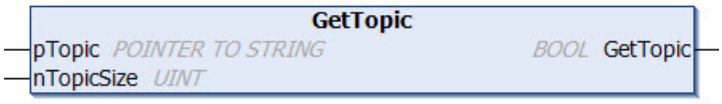
Rückgabewert

Name	Typ	Beschreibung
CompareTopic	BOOL	Ist TRUE, wenn das angegebene Topic mit dem Topic der MQTT-Nachricht im Funktionsbaustein identisch ist.

Ein-/Ausgänge

Name	Typ	Beschreibung
sTopic	STRING	

5.1.2.8.2 GetTopic



Syntax

```
METHOD GetTopic : BOOL
VAR_INPUT
    pTopic     : POINTER TO STRING; // topic buffer
    nTopicSize : UINT; // maximum size of topic buffer in bytes
END_VAR
```

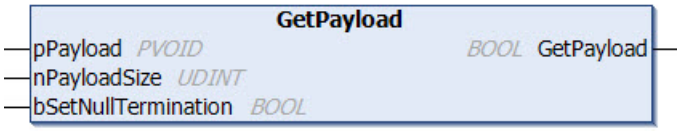
Rückgabewert

Name	Typ	Beschreibung
GetTopic	BOOL	

Eingänge

Name	Typ	Beschreibung
pTopic	POINTER TO STRING	Hier wird die Speicheradresse des Puffers angegeben, in den das Topic kopiert werden soll.
nTopicSize	UINT	Hier wird die maximal zur Verfügung stehende Größe in Bytes des Puffers angegeben.

5.1.2.8.3 GetPayload



Syntax

```
METHOD GetPayload : BOOL
VAR_INPUT
    pPayload           : PVOID; // payload buffer
    nPayloadSize       : UDINT; // maximum size of payload buffer in bytes
    bSetNullTermination: BOOL; // The publisher specifies the kind of payload. If it is a string, it
 could be null terminated or not. Setting this input to TRUE will force a null termination. One more
 byte is required for that.
END_VAR
```

Rückgabewert

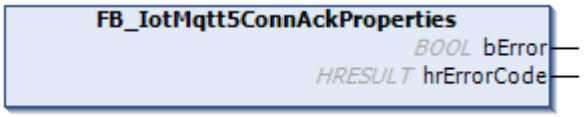
Name	Typ	Beschreibung
GetPayload	BOOL	

Eingänge

Name	Typ	Beschreibung
pPayload	PVOID	Hier wird die Speicheradresse des Puffers, in den das Payload kopiert werden soll angegeben.
nPayloadSize	UDINT	Hier wird die maximal zur Verfügung stehende Größe in Bytes des Puffers angegeben.
bSetNullTermination	BOOL	Erfordert die Art des Payloads eine Null-Terminierung (String), so kann diese hiermit beim Kopiervorgang vorgenommen werden. Hat die Quelle der Nachricht (Publisher) bereits eine Null-Terminierung vorgenommen und diese in der Größenangabe des Payloads berücksichtigt, so ist dies nicht notwendig. Oft bestehen hierzu jedoch keine verlässlichen Informationen.

5.1.2.9 MQTT5 Properties

5.1.2.9.1 FB_IotMqtt5ConnAckProperties



Der Funktionsbaustein ermöglicht den Empfang von Statusinformationen des Message Brokers direkt nach dem Connection Acknowledgement, welcher Teil des Verbindungsaufbaus ist. Hierüber kann der Message Broker dem Client Informationen über potenzielle Einschränkungen mitteilen, zum Beispiel eine maximale Paketgröße, die der Message Broker verarbeiten kann.

Es ist Aufgabe des Clients diese Werte abzufragen und ggf. darauf zu reagieren. Wenn der Client ConnAck Properties empfangen hat und diese am Ausgang bereitstellt, wird dies mittels bPropertiesAvailable=TRUE angezeigt.

Allgemeine Beschreibung der Connection Acknowledgement Properties:

Eigenschaft	Beschreibung
Retain available	Gibt an, ob der Message Broker Retain-Nachrichten unterstützt.
Server KeepAlive	Gibt die vom Server zugewiesene KeepAlive-Zeit an.
Shared subscriptions available	Gibt an, ob der Message Broker Shared Subscriptions unterstützt.
Wildcard subscriptions available	Gibt an, ob der Message Broker Wildcard Subscriptions unterstützt, z.B. Subscriptions auf ein # oder + Topic.
Authentication data	Beinhaltet die Authentifizierungsdaten, abhängig von der verwendeten Authentifizierungsmethode.
Maximum packet size	Gibt die maximale Paketgröße des Control Pakets an. Wenn dieser Wert nicht angegeben ist, dann ist keine Maximalgröße explizit gesetzt.
Maximum QoS	Gibt den maximalen QoS Level an, den der Message Broker unterstützt. Einige Message Broker Implementierungen unterstützen zum Beispiel kein QoS 2.
Reason code	Optionaler Reason Code, welchen der Message Broker im Rahmen des ConnAck-Verfahrens an den Client übermitteln kann. Im Zusammenspiel mit der Server Reference (s.u.) kann der Message Broker dem Client hierüber z.B. mitteilen, dass er temporär oder permanent nicht erreichbar und/oder unter einer neuen Adresse erreichbar ist.
Maximum receive	Der Server verwendet diese Eigenschaft, um die Anzahl an QoS1 und QoS2 Publishes zu limitieren, die er für den Client bereit ist gleichzeitig zu verarbeiten.
Session expiry interval	Der Server kann hierüber den Client informieren, dass er ein anderes Session Expiry Interval verwendet als ursprünglich vom Client angefordert.
Response info	Optionale Rückgabeinformationen vom Message Broker an den Client. Ein Anwendungsfall hierfür kann zum Beispiel sein, dass der Broker dem Client einen Topic-Bereich mitteilt der dem Client zugewiesen wurde und auf den er zugreifen kann.
Maximum topic alias	Gibt den maximalen Wert an, der für einen Topic Alias verwendet werden kann.
Assigned client ID	Gibt die vom Message Broker zugewiesene ClientID zurück, sofern der Client keine eigene ClientID angegeben hat.
Server reference	Optionale (Adress-) Referenz auf einen weiteren Message Broker, z.B. bei Reason Code 0x9C oder 0x9D (Server unavailable, Server has moved).
User Properties	User Properties sind Key/Value Paare, welche an die PublishProperties angehängt werden können. Dies erfolgt mittels des Bausteins FB_IotMqtt5UserProperties [▶ 114]. Die Bedeutung der UserProperties ist nicht Bestandteil der MQTT5 Spezifikation und somit applikationsspezifisch.

Syntax

Definition:

```
FUNCTION_BLOCK FB_IotMqtt5ConnAckProperties EXTENDS FB_IotMqtt5UserProperties [▶ 114]
VAR_OUTPUT
    bError           : BOOL;
    hrErrorCode      : HRESULT;
END_VAR
```

Ausgänge

Name	Typ	Beschreibung
bError	BOOL	Wird TRUE, sobald eine Fehlersituation eintritt.
hrErrorCode	HRESULT	Liefert bei einem gesetzten bError-Ausgang einen Fehlercode. Eine Erläuterung zu den möglichen Fehlercodes befindet sich im Anhang.

Eigenschaften

Name	Typ	Zugriff	Beschreibung
bPropertiesAvailable	BOOL	Get	Gibt an, ob Properties vorhanden sind.
bRetainAvailable	BOOL	Get	Gibt an, ob der Message Broker Retain-Nachrichten unterstützt.
bServerKeepAlive	BOOL	Get	Gibt an, ob vom Server ein anderer KeepAlive-Wert verwendet wird als ursprünglich vom Client angefragt wurde.
bSessionPresent	BOOL	Get	Gibt an, ob der Client schon eine Session mit dem Broker hat.
bSharedSubAvailable	BOOL	Get	Gibt an, ob der Message Broker Shared Subscriptions unterstützt.
bSubIdAvailable	BOOL	Get	Gibt an, ob der Message Broker die Verwendung von Subscription Identifiers unterstützt.
bWildcardSubAvailable	BOOL	Get	Gibt an, ob der Message Broker Wildcard Subscriptions unterstützt.
nAuthDataSize	UINT	Get	Gibt die Größe der Authentication Data an.
nMaxPackateSize	UDINT	Get	Gibt die maximale Paketgröße des Control-Pakets an.
nMaxQoS	BYTE	Get	Gibt den maximalen QoS Level an, den der Message Broker unterstützt.
nReasonCode	BYTE	Get	Gibt einen optionalen Reason Code an, welchen der Server im Rahmen des ConnAck-Verfahrens an den Client übermittelt.
nReceiveMax	UINT	Get	Maximale Anzahl an gleichzeitigen QoS 1 und 2 publishes, die der Server unterstützt.
nResponseInfoSize	UINT	Get	Gibt die Größe der Response-Information zurück.
nSessionExpiryInterval	UDINT	Get	Wert in Sekunden für das vom Server unterstützte Session Expiry Interval.
nTopicAliasMax	UINT	Get	Maximalwert für den Topic Alias, den der Server unterstützt.
sAssignedClientId	STRING	Get	Vom Message Broker zugewiesene Client-ID.
sAuthMethod	STRING	Get	Gibt den Namen der verwendeten Authentifizierungsmethode zurück.
sReasonString	STRING	Get	Leserliche Variante des Reason Code.
sServerReference	STRING	Get	Optionale Server Reference, das Format ist nicht durch die Spezifikation festgelegt.

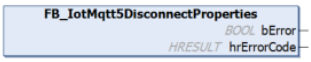
Methoden

Name	Beschreibung
GetAuthData	Gibt die Authentication Data zurück, welche abhängig von der verwendeten Authentication Method ist.
GetResponseInfo	Ermöglicht das Auslesen der Response Information. Hiermit kann der Message Broker dem Client eine Basis für die Erzeugung des Response Topics beim Req/Res-Verfahren [▶ 34] übermitteln. Das Format ist hierbei nicht durch die Spezifikation festgelegt.
SetConnAckProperties	Ermöglicht das Setzen der ConnAck Properties.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4026.0	IPC oder CX (x86, x64, ARM)	Tc3_IotBase (>= v3.4.2.0)

5.1.2.9.2 FB_IotMqtt5DisconnectProperties



Der Funktionsbaustein ermöglicht den Empfang von Informationen zur Verbindungstrennung.

Wenn der Client Disconnect Properties empfangen hat und diese am Ausgang bereitstellt, wird dies mittels bPropertiesAvailable=TRUE angezeigt.

Allgemeine Beschreibung der Disconnect Properties:

Eigenschaft	Beschreibung
Reason Code	Gibt den Grund für den Disconnect an, wie in MQTTv5 Spezifikation Kapitel 3.14.2.1 beschrieben. Wird üblicherweise zur programmatischen Auswertung verwendet.
Reason String	Gibt den Grund für den Disconnect in leserlicher Form an. Wird üblicherweise nicht zur programmatischen Auswertung verwendet.
Session Expiry Interval	Gibt das Ablaufinterval für die Session an (in Sekunden).
Server Reference	Kann vom Message Broker in Zusammenhang mit ReasonCode 0x9C (Use Another Server) oder 0x9D (Server moved) verwendet werden, um einen neuen Server anzugeben. Das Format der Server Reference ist hierbei nicht von der Spezifikation vorgeschrieben und somit abhängig vom verwendeten Message Broker.
User Properties	User Properties sind Key/Value Paare, welche an die PublishProperties angehängt werden können. Dies erfolgt mittels des Bausteins FB_IotMqtt5UserProperties [▶ 114]. Die Bedeutung der UserProperties ist nicht Bestandteil der MQTT5 Spezifikation und somit applikationsspezifisch.

Syntax

Definition:

```
FUNCTION_BLOCK FB_IotMqtt5DisconnectProperties EXTENDS FB_IotMqtt5UserProperties [▶ 114]
VAR_OUTPUT
    bError          : BOOL;
    hrErrorCode     : HRESULT;
END_VAR
```

Ausgänge

Name	Typ	Beschreibung
bError	BOOL	Wird TRUE, sobald eine Fehlersituation eintritt.
hrErrorCode	HRESULT	Liefert bei einem gesetzten bError-Ausgang einen Fehlercode. Eine Erläuterung zu den möglichen Fehlercodes befindet sich im Anhang.

Eigenschaften

Name	Typ	Zugriff	Beschreibung
bPropertiesAvailable	BOOL	Get	Gibt an, ob Properties vorhanden sind.
nReasonCode	BYTE	Get	Gibt den Disconnect Grund als numerischen Wert an.
nSessionExpire	UDINT	Get	Gibt eine Lebensdauer in Sekunden für die Session an.
sReasonString	STRING	Get	Gibt den Disconnect Grund als leserlichen Wert an.
sServerReference	STRING	Get	Gibt bei ReasonCode 0x9C oder 0x9D eine optionale Server Reference an.

Methoden

Name	Beschreibung
SetDisconnectProperties	Ermöglicht das Setzen der Disconnect Properties.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4026.0	IPC oder CX (x86, x64, ARM)	Tc3_IotBase (>= v3.4.2.0)

5.1.2.9.3 FB_IotMqtt5PublishProperties

Der Funktionsbaustein ermöglicht die Definition von verschiedenen Properties, welche beim Versenden einer Nachricht gesetzt werden können. Hiermit können bei der Nachrichtenübermittlung zusätzliche Metadaten an eine Nachricht angehängt werden. Es können die folgenden Metadaten definiert werden:

Allgemeine Beschreibung der Publish Properties:

Eigenschaft	Beschreibung
Content Type	Kann von den Applikationen verwendet werden, um eine inhaltliche Beschreibung des Payloads zu übermitteln.
Topic Alias	Ein Topic-Alias kann verwendet werden, um ein Topic z.B. durch einen Integer-Wert zu repräsentieren anstelle durch einen (potenziell langen) String.
Subscription Identifier	Eine ID um eine Subscription identifizieren zu können.
Message Expiry Interval	Gibt die Lebensdauer einer Nachricht an bis diese vom Message Broker verworfen werden kann.
Payload UTF-8 Indicator	Gibt an, ob die übermittelte Nachricht im UTF-8 Format aufgebaut ist.
User Properties	User Properties sind Key/Value Paare, welche an die PublishProperties angehängt werden können. Dies erfolgt mittels des Bausteins FB_IotMqtt5UserProperties [▶ 114]. Die Bedeutung der UserProperties ist nicht Bestandteil der MQTT5 Spezifikation und somit applikationsspezifisch.

Syntax

Definition:

```
FUNCTION_BLOCK FB_IotMqtt5PublishProperties EXTENDS FB_IotMqtt5UserProperties [▶ 114]
VAR_OUTPUT
    bError          : BOOL;
    hrErrorCode     : HRESULT;
END_VAR
```

Ausgänge

Name	Typ	Beschreibung
bError	BOOL	Wird TRUE, sobald eine Fehlersituation eintritt.
hrErrorCode	HRESULT	Liefert bei einem gesetzten bError-Ausgang einen Fehlercode. Eine Erläuterung zu den möglichen Fehlercodes befindet sich im Anhang.

Eigenschaften

Name	Typ	Zugriff	Beschreibung
bPayloadUtf8	BOOL	Get, Set	Gibt an ob der Nachrichteninhalt UTF-8 formatiert ist.
bTopicAlias	BOOL	Get, Set	Gibt einen Topic-Alias an.
nMsgExpiryInterval	UDINT	Get, Set	Gibt eine Lebensdauer in Sekunden für die Nachricht an.
nSubIdCnt	UINT	Get	Gibt die Anzahl an Subscription-Ids zurück.
pPublishProperties	POINTER TO MqttPublishProperties	Get	Pointer auf ein Objekt vom Typ MqttPublishProperties. Beim Aufruf von FB_IotMqtt5Client.Publish(), sowie Request() und Response() ist es möglich dies direkt zu übergeben.
sContentType	STRING	Get, Set	Gibt einen Content Type für den Nachrichteninhalt an.

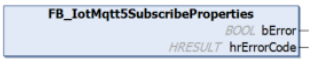
Methoden

Name	Beschreibung
GetSubIds	Gibt die Subscription-IDs zurück.
SetPublishProperties	Setzt die konfigurierten PublishProperties.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4026.0	IPC oder CX (x86, x64, ARM)	Tc3_IotBase (>= v3.4.2.0)

5.1.2.9.4 FB_IotMqtt5SubscribeProperties



Der Funktionsbaustein ermöglicht die Definition von verschiedenen Properties, welche beim Anlegen einer Subscription gesetzt werden können. Hiermit können für eine Subscription zusätzliche Eigenschaften definiert und vom Client an den Message Broker übertragen werden.

Allgemeine Beschreibung der Subscribe Properties:

Eigenschaft	Beschreibung
No Local	Wenn bei Verwendung von MQTTv3 eine Nachricht auf demselben Topic gesendet wird, auf das man sich auch subscribed hat, so empfängt man die gesendete Nachricht nochmals. Durch Verwendung dieses Flags empfängt man bereits gesendete Nachrichten nicht noch einmal.
Retained Message Control	Retain-Nachrichten funktionieren weiterhin wie bei MQTTv3, durch dieses Flag werden jedoch Optionen hinzugefügt, welche definieren was beim Empfang von Retain-Nachrichten passieren soll.
Subscription Identifier	Ein numerischer Wert zur Identifizierung einer Subscription.
User Properties	User Properties sind Key/Value Paare, welche zusätzliche Metadaten transportieren können. Diese werden über den Baustein FB_IotMqtt5UserProperties [▶ 114] verwaltet. Die Bedeutung der UserProperties ist nicht Bestandteil der MQTT5 Spezifikation und somit applikationsspezifisch.

Syntax

Definition:

```
FUNCTION_BLOCK FB_IotMqtt5SubscribeProperties EXTENDS FB_IotMqtt5UserProperties [▶ 114]
VAR_OUTPUT
    bError          : BOOL;
    hrErrorCode     : HRESULT;
END_VAR
```

Ausgänge

Name	Typ	Beschreibung
bError	BOOL	Wird TRUE, sobald eine Fehlersituation eintritt.
hrErrorCode	HRESULT	Liefert bei einem gesetzten bError-Ausgang einen Fehlercode. Eine Erläuterung zu den möglichen Fehlercodes befindet sich im Anhang.

Eigenschaften

Name	Typ	Zugriff	Beschreibung
bNoLocal	BOOL	Get, Set	Gibt an, ob die NoLocal-Eigenschaft gesetzt werden soll, siehe oben.
bRetainAsPublished	BOOL	Get, Set	Gibt an ob das Retain Flag eines Publishers bei einer empfangenen Nachricht gesetzt bleiben soll.
nRetainHandling	BYTE	Get, Set	Gibt an wie der Empfang von Retain-Nachrichten erfolgen soll. 0: Retain-Nachrichten werden vom Broker gesendet wenn sich der Client subscribed. (Default-Verhalten von MQTTv3) 1: Retain-Nachrichten werden nur dann vom Broker gesendet, wenn sich der Client subscribed wenn die Subscription nicht existiert. 2: Retain-Nachrichten werden nicht vom Broker gesendet wenn sich der Client subscribed.
nSubId	UDINT	Get, Set	Numerischer Wert zur optionalen Identifizierung einer Subscription.
pSubscribeProperties	POINTER TO MqttSubscribeProperties	Get	Pointer auf ein Objekt vom Typ MqttSubscribeProperties. Beim Aufruf von FB_IotMqtt5Client.Subscribe() ist es möglich dies direkt zu übergeben.

Methoden

Name	Beschreibung
SetSubscribeProperties	Ermöglicht das Setzen der Subscribe Properties.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4026.0	IPC oder CX (x86, x64, ARM)	Tc3_IotBase (>= v3.4.2.0)

5.1.2.9.5 FB_IotMqtt5UnsubscribeProperties

Der Funktionsbaustein ermöglicht die Definition von verschiedenen Properties, welche beim Abmelden einer Subscription gesetzt werden können. Hiermit können die folgenden Metadaten vom Client an den Message Broker übertragen werden.

Allgemeine Beschreibung der Unsubscribe Properties:

Eigenschaft	Beschreibung
User Properties	User Properties sind Key/Value-Paare, welche zusätzliche Metadaten transportieren können. Diese werden über den Baustein FB_IotMqtt5UserProperties [▶ 114] verwaltet. Die Bedeutung der UserProperties ist nicht Bestandteil der MQTT5 Spezifikation und somit applikationsspezifisch.

Syntax

Definition:

```
FUNCTION_BLOCK FB_IotMqtt5UnsubscribeProperties EXTENDS FB_IotMqtt5UserProperties [▶ 114]
VAR_OUTPUT
    bError          : BOOL;
    hrErrorCode     : HRESULT;
END_VAR
```

Ausgänge

Name	Typ	Beschreibung
bError	BOOL	Wird TRUE, sobald eine Fehlersituation eintritt.
hrErrorCode	HRESULT	Liefert bei einem gesetzten bError-Ausgang einen Fehlercode. Eine Erläuterung zu den möglichen Fehlercodes befindet sich im Anhang.

Eigenschaften

Name	Typ	Zugriff	Beschreibung
pUnsubscribeProperties	POINTER TO MqttUnsubscribeProperties	Get	Pointer auf ein Objekt vom Typ MqttUnsubscribeProperties. Beim Aufruf von FB_IotMqtt5Client.Unsubscribe() ist es möglich dies direkt zu übergeben.

Methoden

Name	Beschreibung
SetUnsubscribeProperties	Ermöglicht das Setzen der Unsubscribe Properties.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4026.0	IPC oder CX (x86, x64, ARM)	Tc3_IotBase (>= v3.4.2.0)

5.1.2.9.6 FB_IotMqtt5UserProperties

Der Funktionsbaustein ermöglicht die Definition und die Handhabung von UserProperties. UserProperties sind Key/Value-Paare, welche optionale Metadaten beschreiben und zum Beispiel bei PublishProperties Verwendung finden können.

Anzahl der UserProperties

Die Verarbeitung von UserProperties im SPS Echtzeit-Task-Zyklus bedarf ja nach Menge einer gewissen Zeitdauer, weshalb diese besser gering zu halten ist.
Die Menge der maximal möglichen UserProperties lässt sich über den Paramater `cMaxMqtt5UserProps` in der Parameterliste [▶ 117] der Bibliothek Tc3_IotBase einstellen.

Syntax

Definition:

```
FUNCTION_BLOCK FB_IotMqtt5UserProperties
VAR_OUTPUT
    bError           : BOOL;
    hrErrorCode      : HRESULT;
END_VAR
```

Ausgänge

Name	Typ	Beschreibung
bError	BOOL	Wird TRUE, sobald eine Fehlersituation eintritt.
hrErrorCode	HRESULT	Liefert bei einem gesetzten bError-Ausgang einen Fehlercode. Eine Erläuterung zu den möglichen Fehlercodes befindet sich im Anhang.

Eigenschaften

Name	Typ	Zugriff	Beschreibung
nUserPropertyCnt	UINT	Get	Gibt die Anzahl der enthaltenen UserProperties an.

Methoden

Name	Beschreibung
AddUserProperty	Fügt ein UserProperty zur Liste hinzu. Das Property wird hierbei durch dessen Name und Value spezifiziert.
ClearUserProperties	Löscht alle in der Liste enthaltenen UserProperties.
GetUserPropertyByIdx	Gibt den Namen und Wert eines UserProperty zurück, welches durch dessen Position (Index) in der Liste angegeben wird.
GetUserPropertyValueByName	Gibt den Wert eines UserProperty zurück, welches durch dessen Namen angegeben wird.
SetUserProperties	Löscht alle in der Liste enthaltenen UserProperties und fügt eine Menge von UserProperties hinzu, welche durch ein Array von Objekten vom Typ MqttUserProperty angegeben werden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4026.0	IPC oder CX (x86, x64, ARM)	Tc3_IotBase (>= v3.4.2.0)

5.1.3 ETcIotMqttClientState

Syntax

```
TYPE ETcIotMqttClientState :
(
MQTT_ERR_SUCCESS            :=0,
MQTT_ERR_NOMEM              :=1,
MQTT_ERR_PROTOCOL           :=2,
MQTT_ERR_INVAL              :=3,
```

```
MQTT_ERR_NO_CONN            :=4,
MQTT_ERR_CONN_REFUSED       :=5,
MQTT_ERR_NOT_FOUND          :=6,
MQTT_ERR_CONN_LOST          :=7,
MQTT_ERR_TLS                :=8,
MQTT_ERR_PAYLOAD_SIZE       :=9,
MQTT_ERR_NOT_SUPPORTED      :=10,
MQTT_ERR_AUTH               :=11,
MQTT_ERR_ACL_DENIED         :=12,
MQTT_ERR_UNKNOWN            :=13,
MQTT_ERR_ERRNO              :=14,
MQTT_ERR_EAI                :=15,
MQTT_ERR_PROXY              :=16,
MQTT_ERR_TLS_CA_NOTFOUND    :=17,
MQTT_ERR_TLS_CERT_NOTFOUND  :=18,
MQTT_ERR_TLS_KEY_NOTFOUND   :=19,
MQTT_ERR_TLS_CA_INVALID     :=20,
MQTT_ERR_TLS_CERT_INVALID   :=21,
MQTT_ERR_TLS_KEY_INVALID    :=22,
MQTT_ERR_TLS_VERIFY_FAIL    :=23,
MQTT_ERR_TLS_SETUP          :=24,
MQTT_ERR_TLS_HANDSHAKE_FAIL :=25,
MQTT_ERR_TLS_CIPHER_INVALID :=26,
MQTT_ERR_TLS_VERSION_INVALID:=27,
MQTT_ERR_TLS_PSK_INVALID    :=28,
MQTT_ERR_TLS_CRL_NOTFOUND   :=29,
MQTT_ERR_TLS_CRL_INVALID    :=30,
MQTT_ERR_FINALIZE_DISCONNECT:=31,
MQTT_ERR_BIND               :=32,
MQTT_ERR_BIND_ADDR_INUSE    :=33,
MQTT_ERR_BIND_ADDR_INVAL    :=34,
MQTT_ERR_CREATE             :=35,
MQTT_ERR_CREATE_TYPE        :=36,
MQTT_ERR_CONN               :=37,
MQTT_ERR_CONN_TIMEDOUT      :=38,
MQTT_ERR_CONN_HOSTUNREACH   :=39,
MQTT_ERR_TLS_CERT_EXPIRED   :=40,
MQTT_ERR_TLS_CN_MISMATCH    :=41
) DINT;
END_TYPE
```

Parameter

Wert	Beschreibung
MQTT_ERR_SUCCESS	
MQTT_ERR_NOMEM	
MQTT_ERR_PROTOCOL	
MQTT_ERR_INVAL	
MQTT_ERR_NO_CONN	
MQTT_ERR_CONN_REFUSED	
MQTT_ERR_NOT_FOUND	
MQTT_ERR_CONN_LOST	
MQTT_ERR_TLS	
MQTT_ERR_PAYLOAD_SIZE	
MQTT_ERR_NOT_SUPPORTED	
MQTT_ERR_AUTH	
MQTT_ERR_ACL_DENIED	
MQTT_ERR_UNKNOWN	
MQTT_ERR_ERRNO	
MQTT_ERR_EAI	
MQTT_ERR_PROXY	
MQTT_ERR_TLS_CA_NOTFOUND	
MQTT_ERR_TLS_CERT_NOTFOUND	
MQTT_ERR_TLS_KEY_NOTFOUND	
MQTT_ERR_TLS_CA_INVALID	
MQTT_ERR_TLS_CERT_INVALID	
MQTT_ERR_TLS_KEY_INVALID	
MQTT_ERR_TLS_VERIFY_FAIL	
MQTT_ERR_TLS_SETUP	
MQTT_ERR_TLS_HANDSHAKE_FAIL	
MQTT_ERR_TLS_CIPHER_INVALID	
MQTT_ERR_TLS_VERSION_INVALID	
MQTT_ERR_TLS_PSK_INVALID	
MQTT_ERR_TLS_CRL_NOTFOUND	
MQTT_ERR_TLS_CRL_INVALID	
MQTT_ERR_FINALIZE_DISCONNECT	
MQTT_ERR_BIND	
MQTT_ERR_BIND_ADDR_INUSE	
MQTT_ERR_BIND_ADDR_INVAL	
MQTT_ERR_CREATE	
MQTT_ERR_CREATE_TYPE	
MQTT_ERR_CONN	
MQTT_ERR_CONN_TIMEDOUT	
MQTT_ERR_CONN_HOSTUNREACH	
MQTT_ERR_TLS_CERT_EXPIRED	
MQTT_ERR_TLS_CN_MISMATCH	

5.1.4 Parameterliste

Parameter, welche die MQTT Message Queue (`FB_IotMqttMessageQueue`, `FB_IotMqtt5MessageQueue`) sowie den für die MQTT Nachrichten verwendeten dynamischen Speicher (TwinCAT Router Speicher) beeinflussen:

Name	Typ	Default-Wert	Beschreibung
cMaxSizeOfMqttMessage	UDINT	102400	Maximale Größe in Bytes von einer MQTT-Nachricht, welche in der Message Queue eingesammelt werden kann. Empfängt der `FB_IotMqttClient` eine zu große MQTT-Nachricht, so wird diese beim Ablageversuch in die `FB_IotMqttMessageQueue` vollständig verworfen. Empfängt der `FB_IotMqtt5Client` eine solche MQTT-Nachricht, so wird diese beim Ablageversuch in die `FB_IotMqtt5MessageQueue` zuerst um alle User Properties reduziert. Sollte dies nicht ausreichen, wird die MQTT-Nachricht vollständig verworfen.
cMaxSizeOfMqttMessageQueue	UDINT	1024000	Maximale Größe in Bytes einer gesamten Message Queue inklusive aller eingesammelten MQTT-Nachrichten. Empfängt der `FB_IotMqttClient` eine neue MQTT-Nachricht womit die Maximalgröße der Message Queue überschritten würde, so wird diese beim Ablageversuch in die `FB_IotMqttMessageQueue` vollständig verworfen. Empfängt der `FB_IotMqtt5Client` eine solche MQTT-Nachricht, so wird mit dem Empfang dieser neuen Nachricht gewartet, bis die Warteschlange Platz bietet.
cMaxEntriesInMqttMessageQueue	UDINT	1000	Maximale Anzahl von Nachrichten in einer Message Queue.

Parameter, welche die Eingänge eines MQTT Clients (`FB_IotMqttClient, FB_IotMqtt5Client`) beeinflussen:

Name	Typ	Default-Wert	Beschreibung
cSizeOfMqttClientClientId	UDINT	256	Definition der STRING-Größe in Bytes, wodurch die maximale Länge der ClientId vorgegeben ist.
cSizeOfMqttClientHostName	UDINT	256	Definition der STRING-Größe in Bytes, wodurch die maximale Länge des Host-Namens vorgegeben ist.
cSizeOfMqttClientTopicPrefix	UDINT	256	Definition der STRING-Größe in Bytes, wodurch die maximale Länge des Topic-Prefix vorgegeben ist.
cSizeOfMqttClientUserName	UDINT	256	Definition der STRING-Größe in Bytes, wodurch die maximale Länge des User-Namens vorgegeben ist.
cSizeOfMqttClientUserPwd	UDINT	256	Definition der STRING-Größe in Bytes, wodurch die maximale Länge des User-Passwortes vorgegeben ist.
cSizeOfMqttWillTopic	UDINT	256	Definition der STRING-Größe in Bytes, wodurch die maximale Länge des Will-Topics vorgegeben ist.

Parameter, welche die MQTT 5 Properties beeinflussen:

Name	Typ	Default-Wert	Beschreibung
cSizeOfMqtt5ContentType	UDINT	256	Definition der STRING-Größe in Bytes, wodurch die maximale Länge des Content-Typen vorgegeben ist.
cSizeOfMqtt5AuthMethod	UDINT	256	Definition der STRING-Größe in Bytes, wodurch die maximale Länge der Authentifizierungsmethode vorgegeben ist.
cSizeOfMqtt5AuthData	UDINT	4096	Maximale Größe in Bytes der Authentifizierungsdaten.
cSizeOfMqtt5ServerReference	UDINT	256	Definition der STRING-Größe in Bytes, wodurch die maximale Länge der Serverreferenz vorgegeben ist.
cSizeOfMqtt5ReasonString	UDINT	256	Definition der STRING-Größe in Bytes, wodurch die maximale Länge des Reason-Strings vorgegeben ist.
cMaxMqtt5UserProps	UINT	20	Maximale Anzahl der UserProperties, welche mit einer MQTT 5 Nachricht empfangen bzw. mit Hilfe von `FB_IotMqtt5UserProperties` versendet werden können. **Die Verarbeitung von UserProperties im SPS Echtzeit-Task-Zyklus bedarf je nach Menge einer gewissen Zeitdauer, weshalb diese besser gering zu halten ist.**

5.2 Tc3_JsonXml

5.2.1 Funktionsbausteine

5.2.1.1 FB_JsonDomParser

Dieser Funktionsblock ist von demselben internen Funktionsbaustein abgeleitet wie der FB_JsonDynDomParser [▶ 189] und bietet somit dasselbe Interface.

Die beiden abgeleiteten Funktionsblöcke unterscheiden sich nur in ihrer internen Speicherverwaltung. Der FB_JsonDomParser ist optimiert für schnelles und effizientes Parsen und Erstellen von JSON-Dokumenten, die nur wenig verändert werden. Für JSON-Dokumente, an denen viele Änderungen (bspw. das zyklische Ändern eines bestimmten Wertes im JSON-Dokument) vorgenommen werden, wird der Funktionsbaustein FB_JsonDynDomParser [▶ 189] empfohlen.

⚠ WARNUNG

Verwendung von Router Speicher

Der Funktionsbaustein zieht bei jeder Änderung, z.B. bei den Methoden SetObject() oder SetJson(), neuen Speicher an. Die verwendete Menge von Router Speicher kann hierdurch nach wiederholten Aktionen stark anwachsen. Dieser allokierte Speicher wird erst durch den Aufruf der Methoden NewDocument [▶ 165]() oder ParseDocument [▶ 166]() wieder freigegeben.

Strings im UTF-8-Format

Die hier verwendeten Variablen vom Typ STRING nutzen das UTF-8-Format. Diese STRING-Formatierung ist üblich bei IoT/MQTT-Kommunikation sowie JSON-Dokumenten.

Um Sonderzeichen und Texte verschiedenster Sprachen empfangen zu können, wird der Zeichensatz in den Bibliotheken Tc3_IotBase und Tc3_JsonXml nicht auf den typischen Zeichensatz vom Datentyp STRING beschränkt. Stattdessen wird der Unicode-Zeichensatz als UTF-8-Format in Verbindung mit dem Datentyp STRING verwendet.

Bei Verwendung des ASCII-Zeichensatzes besteht kein Unterschied zwischen der typischen Formatierung in einem STRING und der UTF-8-Formatierung eines STRING.

Weitere Informationen zum UTF-8-STRING-Format sowie vorhandenen Anzeige- und Konvertierungsmöglichkeiten finden Sie in der Dokumentation der SPS-Bibliothek Tc2_Utilities.

Syntax

```
FUNCTION_BLOCK FB_JsonDomParser
VAR_OUTPUT
  initStatus     : HRESULT;
END_VAR
```

Ausgänge

Name	Typ
initStatus	HRESULT

Methoden

Name	Beschreibung
AddArrayMember [▶ 125]	Fügt ein Array-Member zu einem JSON-Objekt hinzu.
AddBase64Member [▶ 125]	Fügt ein Base64-Member zu einem JSON-Objekt hinzu.
AddBoolMember [▶ 126]	Fügt ein Bool-Member zu einem JSON-Objekt hinzu.
AddDateTimeMember [▶ 127]	Fügt ein DateTime-Member zu einem JSON-Objekt hinzu.
AddDcTimeMember [▶ 128]	Fügt ein DcTime-Member zu einem JSON-Objekt hinzu.
AddDoubleMember [▶ 128]	Fügt ein Double-Member zu einem JSON-Objekt hinzu.
AddFileTimeMember [▶ 129]	Fügt ein FileTime-Member zu einem JSON-Objekt hinzu.
AddHexBinaryMember [▶ 130]	Fügt ein HexBinary-Member zu einem JSON-Objekt hinzu.
AddInt64Member [▶ 130]	Fügt ein Int64-Member zu einem JSON-Objekt hinzu.
AddIntMember [▶ 131]	Fügt ein Int-Member zu einem JSON-Objekt hinzu.
AddJsonMember [▶ 132]	Fügt ein JSON-Member zu einem JSON-Objekt hinzu.
AddNullMember [▶ 132]	Fügt ein NULL-Member zu einem JSON-Objekt hinzu.
AddObjectMember [▶ 133]	Fügt ein Object-Member zu einem JSON-Objekt hinzu.
AddStringMember [▶ 134]	Fügt ein String-Member zu einem JSON-Objekt hinzu.
AddUint64Member [▶ 134]	Fügt ein UInt64-Member zu einem JSON-Objekt hinzu.
AddUintMember [▶ 135]	Fügt ein UInt-Member zu einem JSON-Objekt hinzu.
ArrayBegin [▶ 136]	Liefert das erste Element eines Arrays.
ArrayEnd [▶ 136]	Liefert das letzte Element eines Arrays.
ClearArray [▶ 137]	Löscht den Inhalt eines Arrays.
CopyDocument [▶ 138]	Kopiert den Inhalt des DOM-Speichers in eine Variable vom Datentyp STRING.
CopyJson [▶ 139]	Extrahiert ein JSON-Objekt aus einem Key und speichert dieses in einer Variablen vom Datentyp STRING.
CopyString [▶ 140]	Kopiert den Wert eines Keys in eine Variable vom Datentyp STRING.
FindMember [▶ 141]	Sucht in einem JSON-Dokument nach einem bestimmten Property.
FindMemberPath [▶ 142]	Sucht in einem JSON-Dokument nach einem bestimmten Property (Pfad-spezifisch).
GetArraySize [▶ 143]	Liefert die Anzahl der Elemente in einem JSON-Array.
GetArrayValue [▶ 144]	Liefert den Wert an der aktuellen Iterator-Position eines Arrays.
GetArrayValueByIdx [▶ 144]	Liefert den Wert eines Arrays an einem angegebenen Index.
GetBase64 [▶ 145]	Dekodiert einen Base64-Wert aus einem JSON-Property.
GetBool [▶ 145]	Liefert den Value eines Properties vom Datentyp BOOL.
GetDateTime [▶ 146]	Liefert den Value eines Properties vom Datentyp DATE_AND_TIME.
GetDcTime [▶ 146]	Liefert den Values eines Properties vom Datentyp DCTIME.
GetDocument [▶ 147]	Gibt den Inhalt des DOM-Speichers als Datentyp STRING(255) zurück.
GetDocumentLength [▶ 147]	Gibt die Länge eines JSON-Dokuments im DOM-Speicher zurück.
GetDocumentRoot [▶ 147]	Liefert den Root-Knoten eines JSON-Dokuments im DOM-Speicher.
GetDouble [▶ 148]	Liefert den Value eines Properties vom Datentyp LREAL.
GetFileTime [▶ 148]	Liefert den Value eines Properties vom Datentyp DCTIME.
GetHexBinary [▶ 149]	Dekodiert den HexBinary-Inhalt eines Properties und schreibt diesen an eine bestimmte Speicheradresse,
GetInt [▶ 149]	Liefert den Value eines Properties vom Datentyp DINT.
GetInt64 [▶ 150]	Liefert den Value eines Properties vom Datentyp LINT.
GetJson [▶ 150]	Liefert den Value eines Properties als Datentyp STRING(255) zurück.

Name	Beschreibung
GetJsonLength [▶ 151]	Liefert die Länge eines Properties, wenn dieses ein JSON-Dokument ist.
GetMaxDecimalPlaces [▶ 151]	Liefert die aktuelle Einstellung für MaxDecimalPlaces.
GetMemberName [▶ 151]	Liefert den Namen eines JSON-Property-Members an der Position des aktuellen Iterators,
GetMemberValue [▶ 152]	Liefert den Value eines JSON-Property-Members an der Position des aktuellen Iterators,
GetString [▶ 153]	Liefert den Value eines Properties vom Datentyp STRING(255).
GetStringLength [▶ 153]	Liefert die Länge eines Properties, wenn dessen Value ein String ist.
GetType [▶ 154]	Liefert den Typ eines Property-Values.
GetUint [▶ 154]	Liefert den Value eines Properties vom Datentyp UDINT.
GetUint64 [▶ 155]	Liefert den Value eines Properties vom Datentyp ULINT.
HasMember [▶ 155]	Prüft, ob ein bestimmtes Property im DOM-Speicher vorhanden ist.
IsArray [▶ 156]	Prüft, ob es sich bei einem gegebenen Property um ein Array handelt.
IsBase64 [▶ 156]	Prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp Base64 handelt.
IsBool [▶ 157]	Prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp BOOL handelt.
IsDouble [▶ 157]	Prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp Double (SPS: LREAL) handelt.
IsFalse [▶ 158]	Prüft, ob der Value eines gegebenen Properties FALSE ist.
IsHexBinary [▶ 158]	Prüft, ob der Value eines Properties ein HexBinary-Format hat.
IsInt [▶ 159]	Prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp Integer (SPS: DINT) handelt.
IsInt64 [▶ 159]	Prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp LINT handelt.
IsISO8601TimeFormat [▶ 159]	Prüft, ob es sich bei dem Value eines gegebenen Properties um ein Zeitformat laut ISO8601 handelt.
IsNull [▶ 160]	Prüft, ob es sich bei dem Value eines gegebenen Properties um NULL handelt.
IsNumber [▶ 160]	Prüft, ob es sich bei dem Value eines gegebenen Properties um einen numerischen Wert handelt.
IsObject [▶ 161]	Prüft, ob es sich bei dem gegebenen Property um ein weiteres JSON-Objekt handelt.
IsString [▶ 161]	Prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp STRING handelt.
IsTrue [▶ 162]	Prüft, ob der Wert eines gegebenen Properties TRUE ist.
IsUint [▶ 162]	Prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp UDINT handelt.
IsUint64 [▶ 163]	Prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp ULINT handelt.
LoadDocumentFromFile [▶ 163]	Lädt ein JSON-Dokument aus einer Datei.
MemberBegin [▶ 164]	Liefert das erste Kindelement unterhalb eines JSON-Properties.
MemberEnd [▶ 164]	Liefert das letzte Kindelement unterhalb eines JSON-Properties.
NewDocument [▶ 165]	Erzeugt ein neues, leeres JSON-Dokument im DOM-Speicher.
NextArray [▶ 165]	Liefert das nächste Element in einem Array.
NextMember	Liefert das nächste Property in einem JSON-Dokument.
ParseDocument [▶ 166]	Lädt ein JSON-Objekt zur weiteren Verarbeitung in den DOM-Speicher.
PopbackValue	Löscht das Element am Ende eines Arrays.

Name	Beschreibung
PushbackBase64Value [▶ 166]	Hängt einen Base64-Wert an das Ende eines Arrays an.
PushbackBoolValue [▶ 167]	Hängt einen Base64-Wert an das Ende eines Arrays an.
PushbackDateTimeValue [▶ 168]	Hängt einen Wert vom Datentyp DATE_AND_TIME an das Ende eines Arrays an.
PushbackDcTimeValue [▶ 169]	Hängt einen Wert vom Datentyp DCTIME an das Ende eines Arrays an.
PushbackDoubleValue [▶ 169]	Hängt einen Wert vom Datentyp Double an das Ende eines Arrays an.
PushbackFileTimeValue [▶ 170]	Hängt einen Wert vom Datentyp FILETIME an das Ende eines Arrays an.
PushbackHexBinaryValue [▶ 170]	Hängt einen HexBinary-kodierten Wert an das Ende eines Arrays an.
PushbackInt64Value [▶ 171]	Hängt einen Wert vom Datentyp Int64 an das Ende eines Arrays an.
PushbackIntValue [▶ 171]	Hängt einen Wert vom Datentyp INT an das Ende eines Arrays an.
PushbackJsonValue [▶ 172]	Fügt ein JSON-Dokument zum Ende eines Arrays hinzu.
PushbackNullValue [▶ 173]	Hängt einen NULL-Wert an das Ende eines Arrays an.
PushbackStringValue [▶ 173]	Hängt einen Wert vom Datentyp String an das Ende eines Arrays an.
PushbackUint64Value [▶ 174]	Hängt einen Wert vom Datentyp UInt64 an das Ende eines Arrays an.
PushbackUintValue [▶ 174]	Hängt einen Wert vom Datentyp UInt an das Ende eines Arrays an.
RemoveAllMembers [▶ 175]	Entfernt alle Kindelemente von einem gegebenen Property.
RemoveArray [▶ 175]	Löscht den Wert des aktuellen Array-Iterators.
RemoveMember [▶ 176]	Löscht das Property an dem aktuellen Iterator.
RemoveMemberByName [▶ 177]	Entfernt ein Kindelement von einem gegebenen Property.
SaveDocumentToFile [▶ 177]	Speichert ein JSON-Dokument in einer Datei.
SetArray [▶ 179]	Setzt den Value eines Properties auf den Typ „Array“.
SetBase64 [▶ 179]	Setzt den Value eines Properties auf einen Base64-kodierten Wert.
SetBool [▶ 180]	Setzt den Value eines Properties auf einen Wert vom Datentyp BOOL.
SetDateTime [▶ 180]	Setzt den Value eines Properties auf einen Wert vom Datentyp DATE_AND_TIME.
SetDcTime [▶ 181]	Setzt den Value eines Properties auf einen Wert vom Datentyp DCTIME.
SetDouble [▶ 182]	Setzt den Value eines Properties auf einen Wert vom Datentyp Double.
SetFileTime [▶ 182]	Setzt den Value eines Properties auf einen Wert vom Datentyp FILETIME.
SetHexBinary [▶ 183]	Setzt den Value eines Properties auf einen HexBinary-kodierten Wert.
SetInt [▶ 183]	Setzt den Value eines Properties auf einen Wert vom Datentyp INT
SetInt64 [▶ 184]	Setzt den Value eines Properties auf einen Wert vom Datentyp Int64.
SetJson [▶ 184]	Fügt in den Value eines Properties ein weiteres JSON-Dokument ein.
SetMaxDecimalPlaces [▶ 185]	Setzt die aktuelle Einstellung für MaxDecimalPlaces.
SetNull [▶ 185]	Setzt den Value eines Properties auf den Wert NULL.
SetObject [▶ 186]	Setzt den Value eines Properties auf den Typ „Object“.
SetString [▶ 186]	Setzt den Value eines Properties auf einen Wert vom Datentyp STRING.
SetUint [▶ 187]	Setzt den Value eines Properties auf einen Wert vom Datentyp UInt.
SetUint64 [▶ 188]	Setzt den Value eines Properties auf einen Wert vom Datentyp UInt64.

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4022	x86, x64, ARM	Tc3_JsonXml

5.2.1.1.1 AddArrayMember

Diese Methode fügt ein Array-Member zu einem JSON-Objekt hinzu.

Syntax

```
METHOD AddArrayMember : SJsonValue
VAR_INPUT
  v       : SJsonValue;
END_VAR
VAR_IN_OUT CONSTANT
  member  : STRING;
END_VAR
VAR_INPUT
  reserve : UDINT;
END_VAR
```

Rückgabewert

Name	Typ
AddArrayMember	SJsonValue

Eingänge

Name	Typ
v	SJsonValue
reserve	UDINT

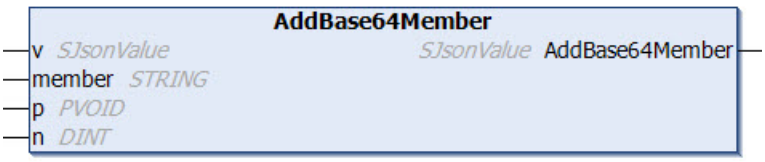
Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonArray := fbJson.AddArrayMember(jsonDoc, 'TestArray', 0);
```

5.2.1.1.2 AddBase64Member



Diese Methode fügt ein Base64-Member zu einem JSON-Objekt hinzu. Als Eingabeparameter kann z. B. eine Struktur adressiert werden. Die entsprechende Base64-Kodierung erfolgt durch die Methode.

Syntax

```
METHOD AddBase64Member : SJsonValue
VAR_INPUT
  v      : SJsonValue;
  p      : PVOID;
  n      : DINT;
END_VAR
VAR_IN_OUT CONSTANT
  member : STRING;
END_VAR
```

Rückgabewert

Name	Typ
AddBase64Member	SJsonValue

Eingänge

Name	Typ
v	SJsonValue
p	PVOID
n	DINT

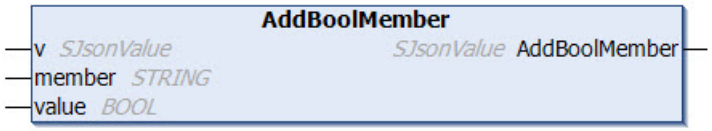
/ Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

```
jsonDoc    := fbJson.ParseDocument(sExistingJsonDocument);
jsonBase64 := fbJson.AddBase64Member(jsonDoc, 'TestBase64', ADR(stStruct), SIZEOF(stStruct));
```

5.2.1.1.3 AddBoolMember



Diese Methode fügt ein Bool-Member zu einem JSON-Objekt hinzu.

Syntax

```
METHOD AddBoolMember : SJsonValue
VAR_INPUT
  v      : SJsonValue;
  value  : BOOL;
END_VAR
VAR_IN_OUT CONSTANT
  member : STRING;
END_VAR
```

Rückgabewert

Name	Typ
AddBoolMember	SJsonValue

Eingänge

Name	Typ
v	SJsonValue
value	BOOL

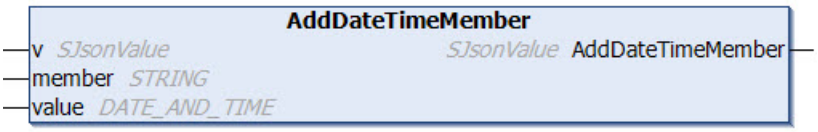
/ Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);
jsonMem := fbJson.AddBoolMember(jsonDoc, 'TestBool', TRUE);
```

5.2.1.1.4 AddDateTimeMember



Diese Methode fügt ein DateTime-Member zu einem JSON-Objekt hinzu.

Syntax

```
METHOD AddDateTimeMember : SJsonValue
VAR_INPUT
  v      : SJsonValue;
  value  : DATE_AND_TIME;
END_VAR
VAR_IN_OUT CONSTANT
  member : STRING;
END_VAR
```

Rückgabewert

Name	Typ
AddDateTimeMember	SJsonValue

Eingänge

Name	Typ
v	SJsonValue
value	DATE_AND_TIME

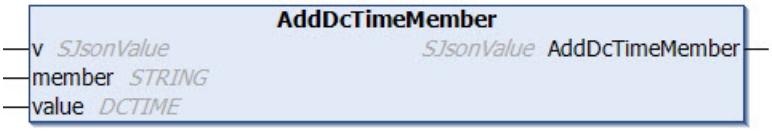
/ Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);
jsonMem := fbJson.AddDateTimeMember(jsonDoc, 'TestDateTime', DT#2018-11-22-12:12);
```

5.2.1.1.5 AddDcTimeMember



Diese Methode fügt ein DcTime-Member zu einem JSON-Objekt hinzu.

Syntax

```
METHOD AddDcTimeMember : SJsonValue
VAR_INPUT
  v        : SJsonValue;
  value    : DCTIME;
END_VAR
VAR_IN_OUT CONSTANT
    member : STRING;
END_VAR
```

Rückgabewert

Name	Typ
AddDcTimeMember	SJsonValue

Eingänge

Name	Typ
v	SJsonValue
value	DCTIME

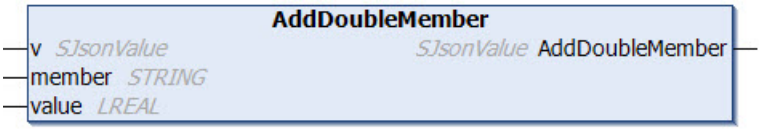
/ Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);
jsonMem := fbJson.AddDcTimeMember(jsonDoc, 'TestDcTime', 1234);
```

5.2.1.1.6 AddDoubleMember



Diese Methode fügt ein Double-Member zu einem JSON-Objekt hinzu.

Syntax

```
METHOD AddDoubleMember : SJsonValue
VAR_INPUT
  v      : SJsonValue;
  value  : LREAL;
END_VAR
VAR_IN_OUT CONSTANT
  member : STRING;
END_VAR
```

Rückgabewert

Name	Typ
AddDoubleMember	SJsonValue

Eingänge

Name	Typ
v	SJsonValue
value	LREAL

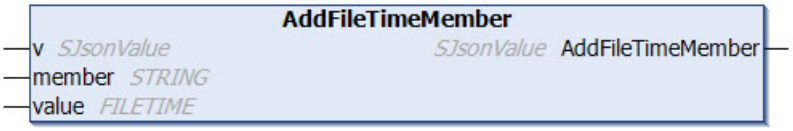
/ Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);
jsonMem := fbJson.AddDoubleMember(jsonDoc, 'TestDouble', 42.42);
```

5.2.1.1.7 AddFileTimeMember



Diese Methode fügt ein FileTime-Member zu einem JSON-Objekt hinzu.

Syntax

```
METHOD AddFileTimeMember : SJsonValue
VAR_INPUT
  v      : SJsonValue;
  value  : FILETIME;
END_VAR
VAR_IN_OUT CONSTANT
  member : STRING;
END_VAR
```

Rückgabewert

Name	Typ
AddFileTimeMember	SJsonValue

Eingänge

Name	Typ
v	SJsonValue
value	FILETIME

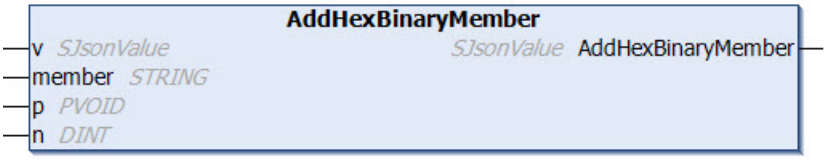
/ Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);
jsonMem := fbJson.AddFileTimeMember(jsonDoc, 'TestFileTime', ftTime);
```

5.2.1.1.8 AddHexBinaryMember



Diese Methode fügt ein HexBinary-Member zu einem JSON-Objekt hinzu.

Syntax

```
METHOD AddHexBinaryMember : SJsonValue
VAR_INPUT
  v       : SJsonValue;
  p       : PVOID;
  n       : DINT;
END_VAR
VAR_IN_OUT CONSTANT
  member : STRING;
END_VAR
```

Rückgabewert

Name	Typ
AddHexBinaryMember	SjsonValue

Eingänge

Name	Typ
v	SjsonValue
p	PVOID
n	DINT

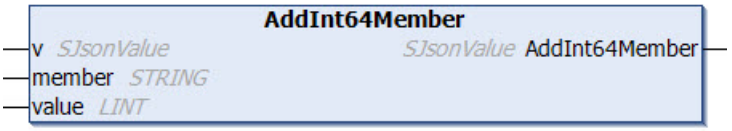
/ Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);
jsonMem := fbJson.AddHexBinaryMember(jsonDoc, ,TestHexBinary', sHexBinary, SIZEOF(sHexBinary));
```

5.2.1.1.9 AddInt64Member



Diese Methode fügt ein Int64-Member zu einem JSON-Objekt hinzu.

Syntax

```
METHOD AddFileTimeMember : SjsonValue
VAR_INPUT
  v       : SjsonValue;
  value   : LINT;
END_VAR
```

```
VAR_IN_OUT CONSTANT
  member : STRING;
END_VAR
```

Rückgabewert

Name	Typ
AddInt64Member	SJsonValue

Eingänge

Name	Typ
v	SJsonValue
value	LINT

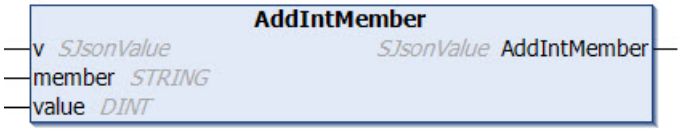
/ Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);
jsonMem := fbJson.AddInt64Member(jsonDoc, 'TestInt64', 42);
```

5.2.1.1.10 AddIntMember



Diese Methode fügt ein Int-Member zu einem JSON-Objekt hinzu.

Syntax

```
METHOD AddIntMember : SJsonValue
VAR_INPUT
  v      : SJsonValue;
  value  : DINT;
END_VAR
VAR_IN_OUT CONSTANT
  member : STRING;
END_VAR
```

Rückgabewert

Name	Typ
AddIntMember	SJsonValue

Eingänge

Name	Typ
v	SJsonValue
value	DINT

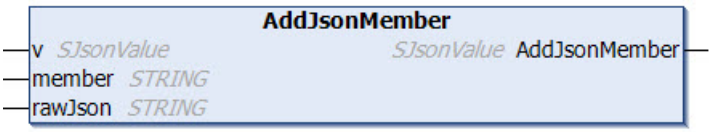
/ Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);
jsonMem := fbJson.AddIntMember(jsonDoc, 'TestInt', 42);
```

5.2.1.1.11 AddJsonMember



Diese Methode fügt ein JSON-Member zu einem JSON-Objekt hinzu.

Syntax

```
METHOD AddJsonMember: SJsonValue
VAR_INPUT
  v       : SJsonValue;
END_VAR
VAR_IN_OUT CONSTANT
  member  : STRING;
  rawJson : STRING;
END_VAR
```

Rückgabewert

Name	Typ
AddJsonMember	SJsonValue

Eingänge

Name	Typ
v	SJsonValue

/ Ein-/Ausgänge

Name	Typ
member	STRING
rawJson	STRING

Beispielaufruf:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);
jsonMem := fbJson.AddJsonMember(jsonDoc, 'TestJson', sJson);
```

5.2.1.1.12 AddNullMember

Diese Methode fügt ein NULL-Member zu einem JSON-Objekt hinzu.

Syntax

```
METHOD AddNullMember : SJsonValue
VAR_INPUT
  v       : SJsonValue;
END_VAR
VAR_IN_OUT CONSTANT
  member : STRING;
END_VAR
```

Rückgabewert

Name	Typ
AddNullMember	SJsonValue

Eingänge

Name	Typ
v	SJsonValue

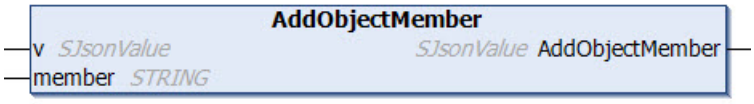
/ Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);
jsonMem := fbJson.AddNullMember(jsonDoc, 'TestJson');
```

5.2.1.1.13 AddObjectMember



Diese Methode fügt ein Object-Member zu einem JSON-Objekt hinzu.

Syntax

```
METHOD AddObjectMember : SJsonValue
VAR_INPUT
 v       : SJsonValue;
END_VAR
VAR_IN_OUT CONSTANT
  member : STRING;
END_VAR
```

Rückgabewert

Name	Typ
AddObjectMember	SJsonValue

Eingänge

Name	Typ
v	SJsonValue

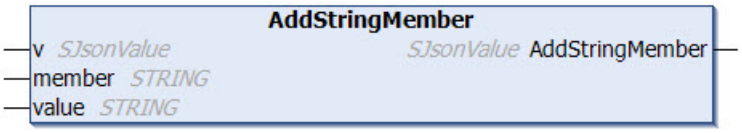
/ Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);
jsonMem := fbJson.AddObjectMember(jsonDoc, 'TestObject');
```

5.2.1.1.14 AddStringMember



Diese Methode fügt ein String-Member zu einem JSON-Objekt hinzu.

Syntax

```
METHOD AddStringMember : SJsonValue
VAR_INPUT
  v      : SJsonValue;
END_VAR
VAR_IN_OUT CONSTANT
  member : STRING;
  value  : STRING;
END_VAR
```

Rückgabewert

Name	Typ
AddStringMember	SJsonValue

Eingänge

Name	Typ
v	SJsonValue

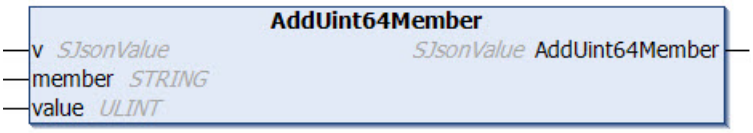
/ Ein-/Ausgänge

Name	Typ
member	STRING
value	STRING

Beispielaufruf:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);
jsonMem := fbJson.AddStringMember(jsonDoc, 'TestString', 'Test');
```

5.2.1.1.15 AddUint64Member



Diese Methode fügt ein UInt64-Member zu einem JSON-Objekt hinzu.

Syntax

```
METHOD AddUint64Member : SJsonValue
VAR_INPUT
  v      : SJsonValue;
  value  : ULINT;
END_VAR
VAR_IN_OUT CONSTANT
  member : STRING;
END_VAR
```

Rückgabewert

Name	Typ
AddUint64Member	SJsonValue

Eingänge

Name	Typ
v	SJsonValue
value	ULINT

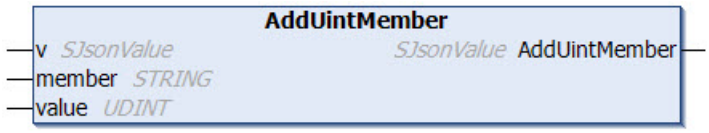
/ Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);
jsonMem := fbJson.AddUint64Member(jsonDoc, 'TestUint64', 42);
```

5.2.1.1.16 AddUintMember



Diese Methode fügt ein UInt-Member zu einem JSON-Objekt hinzu.

Syntax

```
METHOD AddUintMember : SJsonValue
VAR_INPUT
  v      : SJsonValue;
  value  : UDINT;
END_VAR
VAR_IN_OUT CONSTANT
  member : STRING;
END_VAR
```

Rückgabewert

Name	Typ
AddUintMember	SJsonValue

Eingänge

Name	Typ
v	SJsonValue
value	UDINT

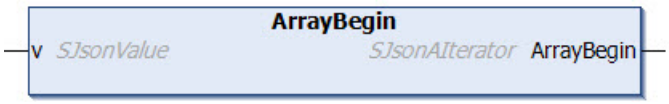
/ Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);
jsonMem := fbJson.AddUintMember(jsonDoc, 'TestUint', 42);
```

5.2.1.1.17 ArrayBegin



Diese Methode liefert das erste Element eines Arrays und kann zusammen mit den Methoden ArrayEnd() und NextArray() zur Iteration durch ein JSON-Array verwendet werden.

Syntax

```
METHOD ArrayBegin : SJsonAIterator
VAR_INPUT
  v : SJsonValue;
END_VAR
```

Rückgabewert

Name	Typ
ArrayBegin	SJsonAIterator

Eingänge

Name	Typ
v	SJsonValue

Beispielaufruf:

```
jsonIterator    := fbJson.ArrayBegin(jsonArray);
jsonIteratorEnd := fbJson.ArrayEnd(jsonArray);
WHILE jsonIterator <> jsonIteratorEnd DO
  sName         := fbJson.GetArrayValue(jsonIterator);
  jsonIterator  := fbJson.NextArray(jsonIterator);
END_WHILE
```

5.2.1.1.18 ArrayEnd

Diese Methode liefert das letzte Element eines Arrays und kann zusammen mit den Methoden ArrayBegin() und NextArray() zur Iteration durch ein JSON-Array verwendet werden.

Syntax

```
METHOD ArrayEnd : SJsonAIterator
VAR_INPUT
  v : SJsonValue;
END_VAR
```

Rückgabewert

Name	Typ
ArrayEnd	SJsonAIterator

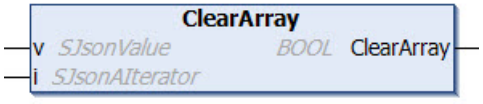
Eingänge

Name	Typ
v	SJsonValue

Beispielaufruf:

```
jsonIterator    := fbJson.ArrayBegin(jsonArray);
jsonIteratorEnd := fbJson.ArrayEnd(jsonArray);
WHILE jsonIterator <> jsonIteratorEnd DO
  sName         := fbJson.GetArrayValue(jsonIterator);
  jsonIterator  := fbJson.NextArray(jsonIterator);
END_WHILE
```

5.2.1.1.19 ClearArray



Diese Methode löscht den Inhalt eines Arrays.

Syntax

```
METHOD ClearArray : BOOL
VAR_INPUT
  v : SJsonValue;
  i : SJsonAIterator;
END_VAR
```

Rückgabewert

Name	Typ
ClearArray	BOOL

Eingänge

Name	Typ
v	SJsonValue
i	SJsonAIterator

Beispielaufruf:

Gegeben sei das folgende JSON-Dokument, das in den DOM-Speicher geladen wird:

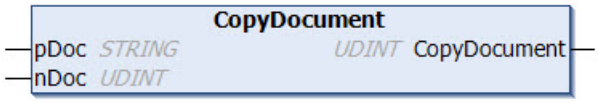
```
sMessage := '{"serialNumber":"123","batteryVoltage":"1547mV","clickType":"SINGLE", "array":
["Hello",2,3]}';
```

Die Werte des JSON-Arrays „array“ sollen gelöscht werden. Zunächst wird das JSON-Dokument iterativ nach dem Property „array“ durchsucht, anschließend werden alle Elemente des Arrays durch Aufruf der Methode ClearArray() gelöscht.

```
jsonDoc              := fbJson.ParseDocument(sMessage);
jsonIterator         := fbJson.MemberBegin(jsonDoc);
jsonIteratorEnd      := fbJson.MemberEnd(jsonDoc);
WHILE jsonIterator <> jsonIteratorEnd DO
  sName := fbJson.GetMemberName(jsonIterator);
  jsonValue          := fbJson.GetMemberValue(jsonIterator);
  IF sName = 'array' THEN
```

```
    jsonArrayIterator := fbJson.ArrayBegin(jsonValue);
    fbJson.ClearArray(jsonValue, jsonArrayIterator);
  END_IF
  jsonIterator        := fbJson.NextMember(jsonIterator);
END_WHILE
```

5.2.1.1.20 CopyDocument



Diese Methode kopiert den Inhalt des DOM-Speichers in eine Variable vom Datentyp STRING, welche eine beliebige Länge haben kann. Als Rückgabewert liefert die Methode die Länge des Strings (inklusive Nullterminierung). Falls der Zielpuffer zu klein ist, wird dieser durch eine Nullterminierung geleert und als Länge 0 zurückgegeben.

Syntax

```
METHOD CopyDocument : UDINT
VAR_INPUT
  nDoc : DINT;
END_VAR
VAR_IN_OUT CONSTANT
  pDoc : STRING;
END_VAR
```

Rückgabewert

Name	Typ
CopyDocument	UDINT

Eingänge

Name	Typ
nDoc	DINT

/ Ein-/Ausgänge

Name	Typ
pDoc	STRING

Beispielaufruf:

```
nLen := fbJson.CopyDocument(sJson, SIZEOF(sJson));
```

5.2.1.1.21 CopyFrom

Syntax

```
METHOD CopyFrom : SJsonValue
VAR_INPUT
  v : SJsonValue;
  w : SJsonValue;
END_VAR
```

Rückgabewert

Name	Typ
CopyFrom	SJsonValue

Eingänge

Name	Typ
v	SJsonValue
w	SJsonValue

5.2.1.1.22 CopyJson

Diese Methode extrahiert ein JSON-Objekt aus einem Key und speichert dieses in einer Variablen vom Datentyp STRING. Dieser STRING kann eine beliebige Länge haben. Als Rückgabewert liefert die Methode die Länge des kopierten JSON-Objekts (inklusive Nullterminierung). Falls der Zielpuffer zu klein ist, wird dieser durch eine Nullterminierung geleert und als Länge 0 zurückgegeben.

Syntax

```
METHOD CopyJson : UDINT
VAR_INPUT
  v    : SJsonValue;
END_VAR
VAR_IN_OUT CONSTANT
  pDoc : STRING;
  nDoc : UDINT;
END_VAR
```

Rückgabewert

Name	Typ
CopyJson	UDINT

Eingänge

Name	Typ
v	SJsonValue

/ Ein-/Ausgänge

Name	Typ
pDoc	STRING
nDoc	STRING

Beispielaufruf:

Gegeben sei das folgende JSON-Dokument, das in den DOM-Speicher geladen wird:

```
sMessage := ' {"serialNumber":"123","meta":{"batteryVoltage":"1547mV","clickType":"SINGLE"}}';
```

Der Wert des JSON-Objekts „meta“ soll extrahiert und in einer Variablen vom Datentyp STRING gespeichert werden. Zunächst wird das JSON-Dokument iterativ nach dem Property „meta“ durchsucht, anschließend wird dessen Wert bzw. Unterobjekt durch Aufruf der Methode CopyJson() extrahiert.

```
jsonDoc         := fbJson.ParseDocument(sMessage);
jsonIterator    := fbJson.MemberBegin(jsonDoc);
jsonIteratorEnd := fbJson.MemberEnd(jsonDoc);
WHILE jsonIterator <> jsonIteratorEnd DO
  sName         := fbJson.GetMemberName(jsonIterator);
  jsonValue     := fbJson.GetMemberValue(jsonIterator);
  IF sName = 'meta' THEN
    fbJson.CopyJson(jsonValue, sString, SIZEOF(sString));
  END_IF
  jsonIterator  := fbJson.NextMember(jsonIterator);
END_WHILE
```

Die Variable sString hat nach diesem Durchlauf folgenden Inhalt:

```
{"batteryVoltage":"1547mV","clickType":"SINGLE"}
```

5.2.1.1.23 CopyString

Diese Methode kopiert den Wert eines Keys in eine Variable vom Datentyp STRING, welche eine beliebige Länge haben kann. Als Rückgabewert liefert die Methode die Länge des kopierten Strings (inklusive Nullterminierung). Falls der Zielpuffer zu klein ist, wird dieser durch eine Nullterminierung geleert und als Länge 0 zurückgegeben.

Syntax

```
METHOD CopyString : UDINT
VAR_INPUT
  v    : SJsonValue;
END_VAR
VAR_IN_OUT CONSTANT
  pStr : STRING;
  nStr : UDINT;
END_VAR
```

Rückgabewert

Name	Typ
CopyString	UDINT

Eingänge

Name	Typ
v	SJsonValue

/ Ein-/Ausgänge

Name	Typ
pStr	STRING
nStr	UDINT

Beispielaufruf:

Gegeben sei das folgende JSON-Dokument, das in den DOM-Speicher geladen wird:

```
sMessage := ' {"serialNumber":"123","batteryVoltage":"1547mV","clickType":"SINGLE"}';
```

Der Wert des Keys „clickType“ soll extrahiert und in einer Variablen vom Datentyp STRING gespeichert werden. Zunächst wird das JSON-Dokument iterativ nach dem Property „clickType“ durchsucht.

```
jsonDoc         := fbJson.ParseDocument(sMessage);
jsonIterator    := fbJson.MemberBegin(jsonDoc);
jsonIteratorEnd := fbJson.MemberEnd(jsonDoc);
```

```
WHILE jsonIterator <> jsonIteratorEnd DO
  sName         := fbJson.GetMemberName(jsonIterator);
  jsonValue     := fbJson.GetMemberValue(jsonIterator);
  IF sName = 'clickType' THEN
    fbJson.CopyString(jsonValue, sString, SIZEOF(sString));
  END_IF
  jsonIterator  := fbJson.NextMember(jsonIterator);
END_WHILE
```

Die Variable sString hat nach diesem Durchlauf folgenden Inhalt:

```
SINGLE
```

5.2.1.1.24 ExceptionRaised

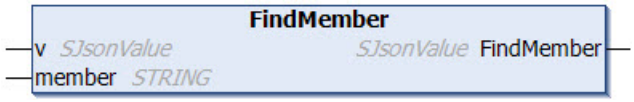
Syntax

```
METHOD ExceptionRaised : BOOL
```

Rückgabewert

Name	Typ
ExceptionRaised	BOOL

5.2.1.1.25 FindMember



Diese Methode sucht in einem JSON-Dokument nach einem bestimmten Property und gibt dieses zurück. Wenn kein entsprechendes Property gefunden wird, wird 0 zurückgegeben.

Syntax

```
METHOD FindMember : SJsonValue
VAR_INPUT
  v      : SJsonValue;
END_VAR
VAR_IN_OUT CONSTANT
  member : STRING;
END_VAR
```

Rückgabewert

Name	Typ
FindMember	SJsonValue

Eingänge

Name	Typ
v	SJsonValue

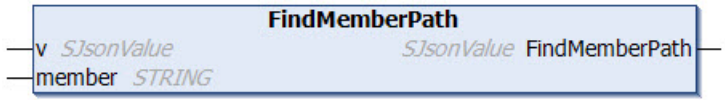
Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

```
jsonProp := fbJson.FindMember(jsonDoc, 'PropertyName');
```

5.2.1.1.26 FindMemberPath



Diese Methode sucht in einem JSON-Dokument nach einem bestimmten Property und gibt dieses zurück. Das Property wird hierbei nach dessen Pfad im Dokument spezifiziert. Wenn kein entsprechendes Property gefunden wird, wird 0 zurückgegeben.

Syntax

```
METHOD FindMemberPath : SJsonValue
VAR_INPUT
  v      : SJsonValue
END_VAR
VAR_IN_OUT CONSTANT
  member : STRING;
END_VAR
```

Rückgabewert

Name	Typ
FindMemberPath	SJsonValue

Eingänge

Name	Typ
v	SJsonValue

/ Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

```
jsonDoc  := fbJson.ParseDocument(sExistingJsonDocument);
jsonProp := fbJson.FindMemberPath(jsonDoc, sPath);
```

Der Zugriff auf verschachtelte Objekte funktioniert nach dem Schema *a/b/c*, um eine Variable in einer JSON-Hierarchie zu finden. Der Aufruf für die Variable *c* des folgenden JSON-Dokuments lautet:

```
jsonProp := fbJson.FindMemberPath(jsonDoc, 'a/b/c');
```

```
{
 "a":{
   "b":{
    "c": 123
       }
     }
}
```

Unterstützung für Arrays

Die Methode unterstützt JSON-Dokumente mit Arrays ab TwinCAT-Version >3.1.4024.35. Mithilfe des Zeichens # kann auf Elemente eines Arrays zugegriffen werden.

```
jsonProp := fbJson.FindMemberPath(jsonDoc, '#1/Third#2');
```

```
[
  {
  "First": 4
  },
  {
```

```
  "Second": 12,
  "Third": [
    1,
    2,
    3
    ],
   "Fourth": {
    "a": true
    }
  },
]
```

Der Beispielaufruf greift auf das zweite Element des äußeren Arrays zu (#1), anschließend auf das dritte Element des Arrays unter dem Unterelement *Third*.

Behandlung von Sonderfällen

Das Einfügen des Zeichens ~ sorgt für die Sonderbehandlung innerhalb eines Pfades. Die folgende Tabelle listet die verschiedenen Möglichkeiten auf.

Ausdruck	Ergebnis	Beispiel
~0	~ wird als Zeichen im String verwendet.	Test/Hallo~0123 wird zu Test/Hallo~123
~1	Der Ausdruck wird durch das Zeichen **/** ersetzt, was nicht als Trenner, sondern als Teil des Strings interpretiert wird.	Test/Hallo~1123 wird Test/Hallo/123
~2	Der Ausdruck wird durch das Zeichen **#** ersetzt, was nicht als Array-Index, sondern als Teil des Strings interpretiert wird.	Test/Hallo~2123 wird zu Test/Hallo#123

5.2.1.1.27 GetArraySize

Diese Methode liefert die Anzahl der Elemente in einem JSON-Array.

Syntax

```
METHOD GetArraySize : UDINT
VAR_INPUT
  v : SJsonValue;
END_VAR
```

Rückgabewert

Name	Typ
GetArraySize	UDINT

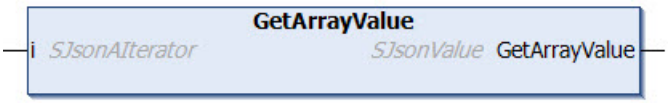
Eingänge

Name	Typ
v	SJsonValue

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonArray := fbJson.FindMember(jsonDoc, 'array');
nSize     := fbJson.GetArraySize(jsonArray);
```

5.2.1.1.28 GetArrayValue



Diese Methode liefert den Wert an der aktuellen Iterator-Position eines Arrays.

Syntax

```
METHOD GetArrayValue : SJsonValue
VAR_INPUT
  i : SJsonAIterator;
END_VAR
```

Rückgabewert

Name	Typ
GetArrayValue	SJsonValue

Eingänge

Name	Typ
i	SJsonAIterator

Beispielaufruf:

```
jsonIterator    := fbJson.ArrayBegin(jsonArray);
jsonIteratorEnd := fbJson.ArrayEnd(jsonArray);
WHILE jsonIterator <> jsonIteratorEnd DO
  sName         := fbJson.GetArrayValue(jsonIterator);
  jsonIterator  := fbJson.NextArray(jsonIterator);
END_WHILE
```

5.2.1.1.29 GetArrayValueByIdx

Diese Methode liefert den Wert eines Arrays an einem angegebenen Index.

Syntax

```
METHOD GetArrayValueByIdx : SJsonValue
VAR_INPUT
  v   : SJsonValue;
  idx : UDINT;
END_VAR
```

Rückgabewert

Name	Typ
GetArrayValueByIdx	SJsonValue

Eingänge

Name	Typ
v	SJsonValue
idx	UDINT

Beispielaufruf:

```
jsonValue := fbJson.GetArrayValueByIdx(jsonArray, 1);
```

5.2.1.1.30 GetBase64

Diese Methode dekodiert einen Base64-Wert aus einem JSON-Property. Wenn sich hinter dem Base64-Wert z. B. der Inhalt einer Datenstruktur befindet, kann der dekodierte Inhalt auch wieder auf eine identische Datenstruktur gelegt werden.

Syntax

```
METHOD GetBase64: DINT
VAR_INPUT
  v : SJsonValue;
  p : PVOID;
  n : DINT;
END_VAR
```

Rückgabewert

Name	Typ
GetBase64	DINT

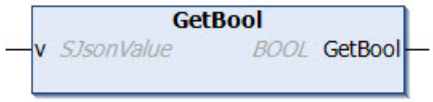
Eingänge

Name	Typ
v	SJsonValue
p	PVOID
n	DINT

Beispielaufruf:

```
jsonDoc    := fbJson.ParseDocument(sExistingJsonDocument);
jsonBase64 := fbJson.FindMember(jsonDoc, 'base64');
nSize      := fbJson.GetBase64(jsonBase64, ADR(stStruct), SIZEOF(stStruct));
```

5.2.1.1.31 GetBool



Diese Methode liefert den Value eines Properties vom Datentyp BOOL.

Syntax

```
METHOD GetBool : BOOL
VAR_INPUT
  v : SJsonValue;
END_VAR
```

Rückgabewert

Name	Typ
GetBool	BOOL

Eingänge

Name	Typ
v	SJsonValue

5.2.1.1.32 GetDateTime

Diese Methode liefert den Value eines Properties vom Datentyp DATE_AND_TIME.

Syntax

```
METHOD GetDateTime : DATE_AND_TIME
VAR_INPUT
  v : SJsonValue;
END_VAR
```

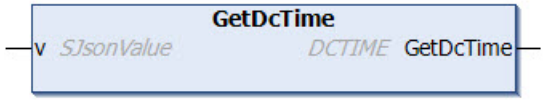
Rückgabewert

Name	Typ
GetDateTime	DATE_AND_TIME

Eingänge

Name	Typ
v	SJsonValue

5.2.1.1.33 GetDcTime



Diese Methode liefert den Values eines Properties vom Datentyp DCTIME.

Syntax

```
METHOD GetDcTime : DCTIME
VAR_INPUT
  v : SJsonValue;
END_VAR
```

Rückgabewert

Name	Typ
GetDcTime	DCTIME

Eingänge

Name	Typ
v	SJsonValue

Beispielaufruf:

```
jsonDoc  := fbJson.ParseDocument(sExistingJsonDocument);
jsonProp := fbJson.FindMember(jsonDoc, 'property');
dcTime   := fbJson.GetDcTime(jsonProp);
```

5.2.1.1.34 GetDocument

Diese Methode gibt den Inhalt des DOM-Speichers als Datentyp STRING(255) zurück. Bei längeren Zeichenfolgen wird von der Methode ein NULL-String zurückgegeben. In diesem Fall muss die Methode CopyDocument [▶ 138]() verwendet werden.

Syntax

```
METHOD GetDocument : STRING(255)
```

Rückgabewert

Name	Typ
GetDocument	STRING(255)

Beispielaufruf:

```
sJson := fbJson.GetDocument();
```

5.2.1.1.35 GetDocumentLength

Diese Methode gibt die Länge eines JSON-Dokuments im DOM-Speicher zurück.

Syntax

```
METHOD GetDocumentLength: UDINT
```

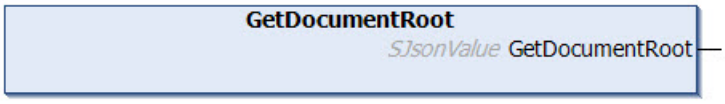
Rückgabewert

Name	Typ
GetDocumentLength	UDINT

Beispielaufruf:

```
nLen := fbJson.GetDocumentLength();
```

5.2.1.1.36 GetDocumentRoot



Diese Methode liefert den Root-Knoten eines JSON-Dokuments im DOM-Speicher.

Syntax

```
METHOD GetDocumentRoot : SJsonValue
```

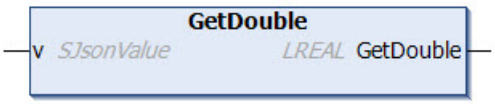
Rückgabewert

Name	Typ
GetDocumentRoot	SJsonValue

Beispielaufruf:

```
jsonRoot := fbJson.GetDocumentRoot();
```

5.2.1.1.37 GetDouble



Diese Methode liefert den Value eines Properties vom Datentyp LREAL.

Syntax

```
METHOD GetDouble : LREAL
VAR_INPUT
  v : SJsonValue;
END_VAR
```

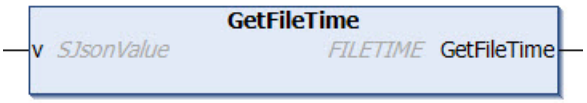
Rückgabewert

Name	Typ
GetDouble	LREAL

Eingänge

Name	Typ
v	SJsonValue

5.2.1.1.38 GetFileTime



Diese Methode liefert den Value eines Properties vom Datentyp DCTIME.

Syntax

```
METHOD GetFileTime : FILETIME
VAR_INPUT
  v : SJsonValue;
END_VAR
```

Rückgabewert

Name	Typ
GetFileTime	FILETIME

Eingänge

Name	Typ
v	SJsonValue

Beispielaufruf:

```
jsonDoc  := fbJson.ParseDocument(sExistingJsonDocument);
jsonProp := fbJson.FindMember(jsonDoc, 'property');
fileTime := fbJson.GetFileTime(jsonProp);
```

5.2.1.1.39 GetHexBinary

Diese Methode dekodiert den HexBinary-Inhalt eines Properties und schreibt diesen an eine bestimmte Speicheradresse, z. B. in eine Datenstruktur.

Syntax

```
METHOD GetHexBinary : DINT
VAR_INPUT
  v : SJsonValue;
  p : PVOID;
  n : DINT;
END_VAR
```

Rückgabewert

Name	Typ
GetHexBinary	DINT

Eingänge

Name	Typ
v	SJsonValue
p	PVOID
n	DINT

Beispielaufruf:

```
jsonDoc  := fbJson.ParseDocument(sExistingJsonDocument);
jsonProp := fbJson.FindMember(jsonDoc, 'property');
nLen     := fbJson.GetHexBinary(jsonProp, ADR(stStruct), SIZEOF(stStruct));
```

5.2.1.1.40 GetInt

Diese Methode liefert den Value eines Properties vom Datentyp DINT.

Syntax

```
METHOD GetInt : DINT
VAR_INPUT
  v : SJsonValue;
END_VAR
```

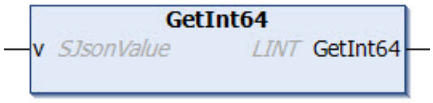
Rückgabewert

Name	Typ
GetInt	DINT

Eingänge

Name	Typ
v	SJsonValue

5.2.1.1.41 GetInt64



Diese Methode liefert den Value eines Properties vom Datentyp LINT.

Syntax

```
METHOD GetInt64 : LINT
VAR_INPUT
  v : SJsonValue;
END_VAR
```

Rückgabewert

Name	Typ
GetInt64	LINT

Eingänge

Name	Typ
v	SJsonValue

5.2.1.1.42 GetJson

Diese Methode liefert den Value eines Properties als Datentyp STRING(255) zurück, wenn dieser selbst wieder ein JSON-Dokument ist. Bei längeren Zeichenfolgen wird von der Methode ein NULL-String zurückgegeben. In diesem Fall muss die Methode CopyJson [▶ 139]() verwendet werden.

Syntax

```
METHOD GetJson : STRING(255)
VAR_INPUT
  v : SJsonValue;
END_VAR
```

Rückgabewert

Name	Typ
GetJson	STRING(255)

Eingänge

Name	Typ
v	SJsonValue

Beispielaufruf:

```
jsonDoc  := fbJson.ParseDocument(sExistingJsonDocument);
jsonProp := fbJson.FindMember(jsonDoc, 'property');
sJson    := fbJson.GetJson(jsonProp);
```

5.2.1.1.43 GetJsonLength

Diese Methode liefert die Länge eines Properties, wenn dieses ein JSON-Dokument ist.

Syntax

```
METHOD GetJsonLength : UDINT
VAR_INPUT
  v : SJsonValue;
END_VAR
```

Rückgabewert

Name	Typ
GetJsonLength	UDINT

Eingänge

Name	Typ
v	SJsonValue

Beispielaufruf:

```
jsonDoc  := fbJson.ParseDocument(sExistingJsonDocument);
jsonProp := fbJson.FindMember(jsonDoc, 'property');
nLen     := fbJson.GetJsonLength(jsonProp);
```

5.2.1.1.44 GetMaxDecimalPlaces

Diese Methode liefert die aktuelle Einstellung für MaxDecimalPlaces. Dies beeinflusst die Anzahl der Dezimalstellen bei Fließkommazahlen.

Syntax

```
METHOD GetMaxDecimalPlaces : DINT
```

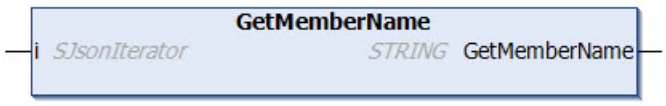
Rückgabewert

Name	Typ
GetMaxDecimalPlaces	DINT

Beispielaufruf:

```
nDec := fbJson.GetMaxDecimalPlaces();
```

5.2.1.1.45 GetMemberName



Diese Methode liefert den Namen eines JSON-Property-Members an der Position des aktuellen Iterators, z. B. während der Iteration durch die Kindelemente eines JSON-Properties mit den Methoden MemberBegin(), MemberEnd() und NextMember().

Syntax

```
METHOD GetMemberName : STRING(255)
VAR_INPUT
  i : SJsonIterator;
END_VAR
```

Rückgabewert

Name	Typ
GetMemberName	STRING(255)

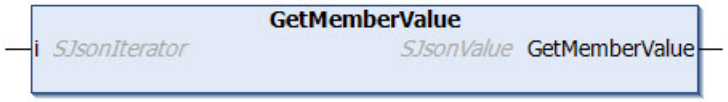
Eingänge

Name	Typ
i	SJsonIterator

Beispielaufruf:

```
jsonDoc         := fbJson.ParseDocument(sExistingJsonDocument);
jsonIterator    := fbJson.MemberBegin(jsonDoc);
jsonIteratorEnd := fbJson.MemberEnd(jsonDoc);
WHILE jsonIterator <> jsonIteratorEnd DO
  sName         := fbJson.GetMemberName(jsonIterator);
  jsonIterator  := fbJson.NextMember(jsonIterator);
END_WHILE
```

5.2.1.1.46 GetMemberValue



Diese Methode liefert den Value eines JSON-Property-Members an der Position des aktuellen Iterators, z.B. während der Iteration durch die Kindelemente eines JSON-Properties mit den Methoden MemberBegin(), MemberEnd() und NextMember().

Syntax

```
METHOD GetMemberValue : SJsonValue
VAR_INPUT
  i : SJsonIterator;
END_VAR
```

Rückgabewert

Name	Typ
GetMemberValue	SJsonValue

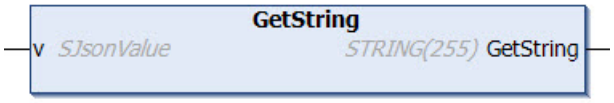
Eingänge

Name	Typ
i	SJsonIterator

Beispielaufruf:

```
jsonDoc         := fbJson.ParseDocument(sExistingJsonDocument);
jsonIterator    := fbJson.MemberBegin(jsonDoc);
jsonIteratorEnd := fbJson.MemberEnd(jsonDoc);
WHILE jsonIterator <> jsonIteratorEnd DO
  jsonValue     := fbJson.GetMemberValue(jsonIterator);
  jsonIterator  := fbJson.NextMember(jsonIterator);
END_WHILE
```

5.2.1.1.47 GetString



Diese Methode liefert den Value eines Properties vom Datentyp STRING(255). Bei längeren Zeichenfolgen wird von der Methode ein NULL-String zurückgegeben. In diesem Fall muss die Methode CopyString [▶ 140] () verwendet werden.

Syntax

```
METHOD GetString : STRING(255)
VAR_INPUT
  v : SJsonValue;
END_VAR
```

Rückgabewert

Name	Typ
GetString	STRING(255)

Eingänge

Name	Typ
v	SJsonValue

5.2.1.1.48 GetStringLength

Diese Methode liefert die Länge eines Properties, wenn dessen Value ein String ist.

Syntax

```
METHOD GetStringLength : UDINT
VAR_INPUT
  v : SJsonValue
END_VAR
```

Rückgabewert

Name	Typ
GetStringLength	UDINT

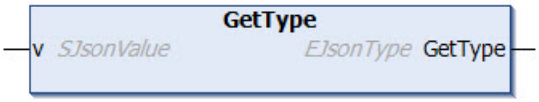
Eingänge

Name	Typ
v	SJsonValue

Beispielaufruf:

```
jsonDoc  := fbJson.ParseDocument(sExistingJsonDocument);
jsonProp := fbJson.FindMember(jsonDoc, 'property');
nLen     := fbJson.GetStringLength(jsonProp);
```

5.2.1.1.49 GetType



Diese Methode liefert den Typ eines Property-Values. Der Rückgabewert kann hierbei einen der Werte des Enums EJsonType annehmen.

Syntax

```
METHOD GetType : EJsonType
VAR_INPUT
  v            : SJsonValue
END_VAR

TYPE EJsonType :
(
  eNullType   := 0,
  eFalseType  := 1,
  eTrueType   := 2,
  eObjectType := 3,
  eArrayType  := 4,
  eStringType := 5,
  eNumberType := 6
) DINT;
```

Rückgabewert

Name	Typ
GetType	EJsonType

Eingänge

Name	Typ
v	SJsonValue

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonProp  := fbJson.FindMember(jsonDoc, 'property');
eJsonType := fbJson.GetType(jsonProp);
```

5.2.1.1.50 GetUint

Diese Methode liefert den Value eines Properties vom Datentyp UDINT.

Syntax

```
METHOD GetUint : UDINT
VAR_INPUT
  v : SJsonValue;
END_VAR
```

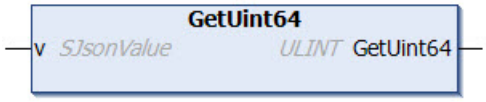
Rückgabewert

Name	Typ
GetUint	UDINT

Eingänge

Name	Typ
v	SJsonValue

5.2.1.1.51 GetUint64



Diese Methode liefert den Value eines Properties vom Datentyp ULINT.

Syntax

```
METHOD GetUint64 : ULINT
VAR_INPUT
  v : SJsonValue;
END_VAR
```

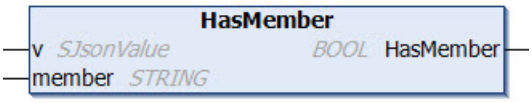
Rückgabewert

Name	Typ
GetUint64	ULINT

Eingänge

Name	Typ
v	SJsonValue

5.2.1.1.52 HasMember



Diese Methode prüft, ob ein bestimmtes Property im DOM-Speicher vorhanden ist. Wenn das Property vorhanden ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

```
METHOD HasMember : BOOL
VAR_INPUT
  v      : SJsonValue;
END_VAR
VAR_IN_OUT CONSTANT
  member : STRING;
END_VAR
```

Rückgabewert

Name	Typ
HasMember	BOOL

Eingänge

Name	Typ
v	SJsonValue

/ Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

```
bHasMember := fbJson.HasMember(jsonDoc, 'PropertyName');
```

5.2.1.1.53 IsArray

Diese Methode prüft, ob es sich bei einem gegebenen Property um ein Array handelt. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

```
METHOD IsArray : BOOL
VAR_INPUT
  v : SJsonValue;
END_VAR
```

Rückgabewert

Name	Typ
IsArray	BOOL

Eingänge

Name	Typ
v	SJsonValue

5.2.1.1.54 IsBase64

Diese Methode prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp Base64 handelt. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

```
METHOD IsBase64 : BOOL
VAR_INPUT
  v : SJsonValue;
END_VAR
```

Rückgabewert

Name	Typ
IsBase64	BOOL

Eingänge

Name	Typ
v	SJsonValue

5.2.1.1.55 IsBool

Diese Methode prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp BOOL handelt. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

```
METHOD IsBool : BOOL
VAR_INPUT
  v : SJsonValue;
END_VAR
```

Rückgabewert

Name	Typ
IsBool	BOOL

Eingänge

Name	Typ
v	SJsonValue

5.2.1.1.56 IsDouble

Diese Methode prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp Double (SPS: LREAL) handelt. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

```
METHOD IsDouble : BOOL
VAR_INPUT
  v : SJsonValue;
END_VAR
```

Rückgabewert

Name	Typ
IsDouble	BOOL

Eingänge

Name	Typ
v	SJsonValue

5.2.1.1.57 IsFalse

Diese Methode prüft, ob der Value eines gegebenen Properties FALSE ist. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

```
METHOD IsFalse : BOOL
VAR_INPUT
  v : SJsonValue;
END_VAR
```

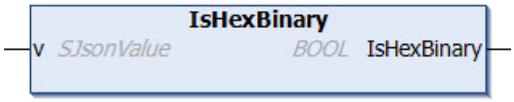
Rückgabewert

Name	Typ
IsFalse	BOOL

Eingänge

Name	Typ
v	SJsonValue

5.2.1.1.58 IsHexBinary



Diese Methode prüft, ob der Value eines Properties ein HexBinary-Format hat. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

```
METHOD IsHexBinary: BOOL
VAR_INPUT
  v : SJsonValue
END_VAR
```

Rückgabewert

Name	Typ
IsHexBinary	BOOL

Eingänge

Name	Typ
v	SJsonValue

Beispielaufruf:

```
jsonDoc  := fbJson.ParseDocument(sExistingJsonDocument);
jsonProp := fbJson.FindMember(jsonDoc, 'property');
bRet     := fbJson.IsHexBinary(jsonProp);
```

5.2.1.1.59 IsInt

Diese Methode prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp Integer (SPS: DINT) handelt. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

```
METHOD IsInt : BOOL
VAR_INPUT
  v : SJsonValue;
END_VAR
```

Rückgabewert

Name	Typ
IsInt	BOOL

Eingänge

Name	Typ
v	SJsonValue

5.2.1.1.60 IsInt64

Diese Methode prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp LINT handelt. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

```
METHOD IsInt64 : BOOL
VAR_INPUT
  v : SJsonValue;
END_VAR
```

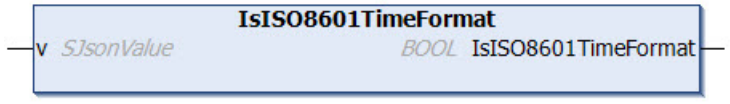
Rückgabewert

Name	Typ
IsInt64	BOOL

Eingänge

Name	Typ
v	SJsonValue

5.2.1.1.61 IsISO8601TimeFormat



Diese Methode prüft, ob es sich bei dem Value eines gegebenen Properties um ein Zeitformat laut ISO8601 handelt. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

```
METHOD IsISO8601TimeFormat : BOOL
VAR_INPUT
  v : SJsonValue;
END_VAR
```

Rückgabewert

Name	Typ
IsISO8601TimeFormat	BOOL

Eingänge

Name	Typ
v	SJsonValue

5.2.1.1.62 IsNull

Diese Methode prüft, ob es sich bei dem Value eines gegebenen Properties um NULL handelt. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

```
METHOD IsNull : BOOL
VAR_INPUT
  v : SJsonValue;
END_VAR
```

Rückgabewert

Name	Typ
IsNull	BOOL

Eingänge

Name	Typ
v	SJsonValue

5.2.1.1.63 IsNumber

Diese Methode prüft, ob es sich bei dem Value eines gegebenen Properties um einen numerischen Wert handelt. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

```
METHOD IsNumber : BOOL
VAR_INPUT
  v : SJsonValue;
END_VAR
```

Rückgabewert

Name	Typ
IsNumber	BOOL

Eingänge

Name	Typ
v	SJsonValue

5.2.1.1.64 IsObject

Diese Methode prüft, ob es sich bei dem gegebenen Property um ein weiteres JSON-Objekt handelt. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

```
METHOD IsObject : BOOL
VAR_INPUT
  v : SJsonValue;
END_VAR
```

Rückgabewert

Name	Typ
IsObject	BOOL

Eingänge

Name	Typ
v	SJsonValue

5.2.1.1.65 IsString

Diese Methode prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp STRING handelt. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

```
METHOD IsString : BOOL
VAR_INPUT
  v : SJsonValue;
END_VAR
```

Rückgabewert

Name	Typ
IsString	BOOL

Eingänge

Name	Typ
v	SJsonValue

5.2.1.1.66 IsTrue

Diese Methode prüft, ob der Wert eines gegebenen Properties TRUE ist. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

```
METHOD IsTrue : BOOL
VAR_INPUT
  v : SJsonValue;
END_VAR
```

Rückgabewert

Name	Typ
IsTrue	BOOL

Eingänge

Name	Typ
v	SJsonValue

5.2.1.1.67 IsUint

Diese Methode prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp UDINT handelt. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

```
METHOD IsUint : BOOL
VAR_INPUT
  v : SJsonValue;
END_VAR
```

Rückgabewert

Name	Typ
IsUint	BOOL

Eingänge

Name	Typ
v	SJsonValue

5.2.1.1.68 IsUint64

Diese Methode prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp ULINT handelt. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

```
METHOD IsUint64 : BOOL
VAR_INPUT
  v : SJsonValue;
END_VAR
```

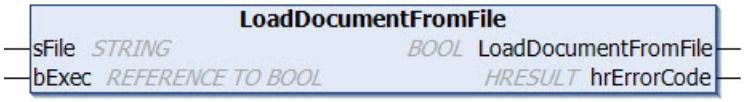
Rückgabewert

Name	Typ
IsUint64	BOOL

Eingänge

Name	Typ
v	SJsonValue

5.2.1.1.69 LoadDocumentFromFile



Diese Methode lädt ein JSON-Dokument aus einer Datei.

Eine steigende Flanke am Eingangsparameter bExec triggert hierbei den Ladevorgang. Der asynchrone Prozess ist beendet, sobald die Referenz bExec von der Methode aus zurück auf FALSE gesetzt wird. Mit Beendigung des Prozesses zeigt der Rückgabewert der Methode für einen Aufruf lang an, ob das Laden der Datei erfolgreich (TRUE) oder fehlgeschlagen (FALSE) war.

Syntax

```
METHOD LoadDocumentFromFile : BOOL
VAR_IN_OUT CONSTANT
  sFile      : STRING;
END_VAR
VAR_INPUT
  bExec      : REFERENCE TO BOOL;
END_VAR
VAR_OUTPUT
  hrErrorCode: HRESULT;
END_VAR
```

Rückgabewert

Name	Typ
LoadDocumentFromFile	BOOL

Eingänge

Name	Typ
bExec	REFERENCE TO BOOL

/ Ein-/Ausgänge

Name	Typ
sFile	STRING

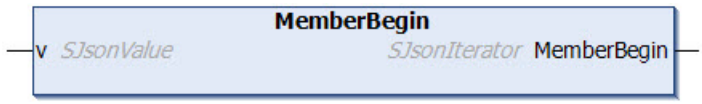
Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

```
IF bLoad THEN
  bLoaded := fbJson.LoadDocumentFromFile(sFile, bLoad);
END_IF
```

5.2.1.1.70 MemberBegin



Diese Methode liefert das erste Kindelemente unterhalb eines JSON-Properties und kann zusammen mit den Methoden MemberEnd() und NextMember() zur Iteration durch ein JSON-Property verwendet werden.

Syntax

```
METHOD MemberBegin : SJsonIterator
VAR_INPUT
  v : SJsonValue;
END_VAR
```

Rückgabewert

Name	Typ
MemberBegin	SJsonIterator

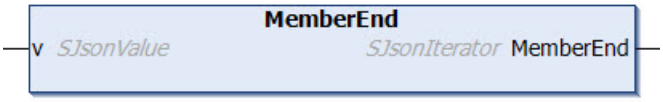
Eingänge

Name	Typ
v	SJsonValue

Beispielaufruf:

```
jsonDoc         := fbJson.ParseDocument(sExistingJsonDocument);
jsonIterator    := fbJson.MemberBegin(jsonDoc);
jsonIteratorEnd := fbJson.MemberEnd(jsonDoc);
WHILE jsonIterator <> jsonIteratorEnd DO
  sName         := fbJson.GetMemberName(jsonIterator);
  jsonIterator  := fbJson.NextMember(jsonIterator);
END_WHILE
```

5.2.1.1.71 MemberEnd



Diese Methode liefert das letzte Kindelement unterhalb eines JSON-Properties und kann zusammen mit den Methoden MemberBegin() und NextMember() zur Iteration durch ein JSON-Property verwendet werden.

Syntax

```
METHOD MemberEnd : SJsonIterator
VAR_INPUT
  v : SJsonValue;
END_VAR
```

Rückgabewert

Name	Typ
MemberEnd	SJsonIterator

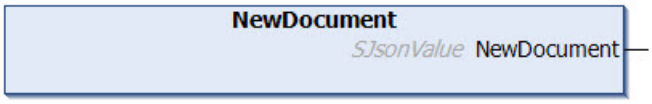
Eingänge

Name	Typ
v	SJsonValue

Beispielaufruf:

```
jsonDoc         := fbJson.ParseDocument(sExistingJsonDocument);
jsonIterator    := fbJson.MemberBegin(jsonDoc);
jsonIteratorEnd := fbJson.MemberEnd(jsonDoc);
WHILE jsonIterator <> jsonIteratorEnd DO
  sName         := fbJson.GetMemberName(jsonIterator);
  jsonIterator  := fbJson.NextMember(jsonIterator);
END_WHILE
```

5.2.1.1.72 NewDocument



Diese Methode erzeugt ein neues, leeres JSON-Dokument im DOM-Speicher.

Syntax

```
METHOD NewDocument : SJsonValue
```

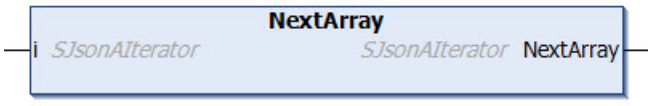
Beispielaufruf:

```
jsonDoc := fbJson.NewDocument();
```

Rückgabewert

Name	Typ
NewDocument	SJsonValue

5.2.1.1.73 NextArray



Syntax

```
METHOD NextArray : SJsonAIterator
VAR_INPUT
  v : SJsonAIterator;
END_VAR
```

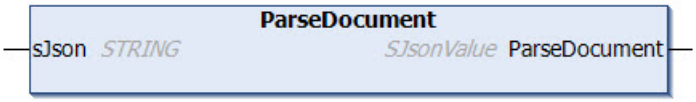
Rückgabewert

Name	Typ
NextArray	SJsonAlterator

Eingänge

Name	Typ
i	SJsonAlterator

5.2.1.1.74 ParseDocument



Diese Methode lädt ein JSON-Objekt zur weiteren Verarbeitung in den DOM-Speicher. Das JSON-Objekt liegt hierbei als String vor und wird als Eingang an die Methode übergeben. Eine Referenz zum JSON-Dokument im DOM-Speicher wird an den Aufrufer zurückgegeben.

Syntax

```
METHOD ParseDocument : SJsonValue
VAR_IN_OUT CONSTANT
  sJson : STRING;
END_VAR
```

Rückgabewert

Name	Typ
ParseDocument	SJsonValue

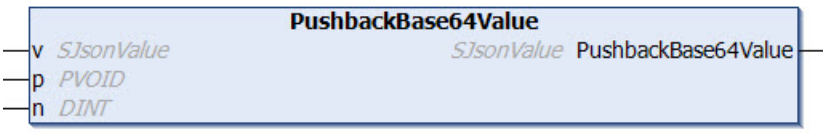
Eingänge

Name	Typ
sJson	STRING

Beispielaufruf:

```
jsonDoc := fbJson.ParseDocument(sJsonString);
```

5.2.1.1.75 PushbackBase64Value



Diese Methode hängt einen Base64-Wert an das Ende eines Arrays an. Als Eingabeparameter kann z. B. eine Struktur adressiert werden. Die entsprechende Base64-Kodierung erfolgt durch die Methode.

Syntax

```
METHOD PushbackBase64Value : SJsonValue
VAR_INPUT
  v : SJsonValue;
  p : PVOID;
  n : DINT;
END_VAR
```

Rückgabewert

Name	Typ
PushbackBase64Value	SJsonValue

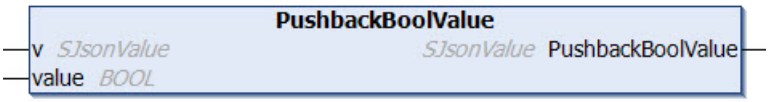
Eingänge

Name	Typ
v	sJsonValue
p	PVOID
n	DINT

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonArray := fbJson.FindMember(jsonDoc, 'array');
jsonValue := fbJson.PushbackBase64Value(jsonArray, ADR(stStruct), SIZEOF(stStruct));
```

5.2.1.1.76 PushbackBoolValue



Diese Methode hängt einen Wert vom Datentyp BOOL an das Ende eines Arrays an.

Syntax

```
METHOD PushbackBoolValue : SJsonValue
VAR_INPUT
  v     : SJsonValue;
  value : BOOL;
END_VAR
```

Rückgabewert

Name	Typ
PushbackBoolValue	SJsonValue

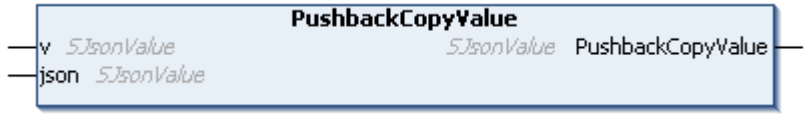
Eingänge

Name	Typ
v	sJsonValue
value	BOOL

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonArray := fbJson.FindMember(jsonDoc, 'array');
jsonValue := fbJson.PushbackBoolValue(jsonArray, TRUE);
```

5.2.1.1.77 PushbackCopyValue



Diese Methode fügt ein JSON-Dokument aus dem Speicher eines FB_JsonDomParser [▶ 119] zum Ende eines Arrays hinzu.

Syntax

```
METHOD PushbackCopyValue : SJsonValue
VAR_INPUT
  v       : SJsonValue;
  json    : SJsonValue;
END_VAR
```

Rückgabewert

Name	Typ
PushbackCopyValue	SJsonValue

Eingänge

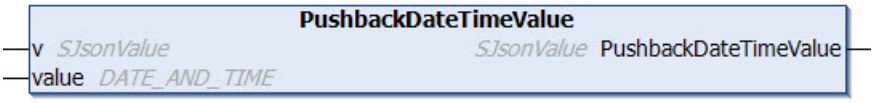
Name	Typ
v	SJsonValue
json	SJsonValue

Beispielaufruf:

```
jsonCopy:=fbJsonCreate.NewDocument();
fbJsonCreate.AddIntMember(jsonCopy,'a',1);
fbJsonCreate.AddIntMember(jsonCopy,'b',2);

jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonArray := fbJson.FindMember(jsonDoc, 'array');
jsonValue := fbJson.PushbackCopyValue(jsonArray, jsonCopy);
```

5.2.1.1.78 PushbackDateTimeValue



Diese Methode hängt einen Wert vom Datentyp DATE_AND_TIME an das Ende eines Arrays an.

Syntax

```
METHOD PushbackDateTimeValue : SJsonValue
VAR_INPUT
  v     : SJsonValue;
  value : DATE_AND_TIME;
END_VAR
```

Rückgabewert

Name	Typ
PushbackDateTimeValue	SJsonValue

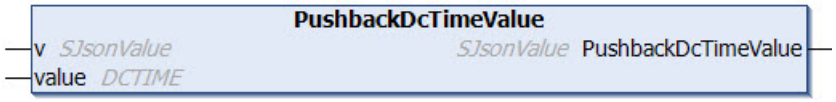
Eingänge

Name	Typ
v	sJsonValue
value	DATE_AND_TIME

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonArray := fbJson.FindMember(jsonDoc, 'array');
jsonValue := fbJson.PushbackDateTimeValue(jsonArray, dtTime);
```

5.2.1.1.79 PushbackDcTimeValue



Diese Methode hängt einen Wert vom Datentyp DCTIME an das Ende eines Arrays an.

Syntax

```
METHOD PushbackDcTimeValue : SJsonValue
VAR_INPUT
  v     : SJsonValue;
  value : DCTIME;
END_VAR
```

Rückgabewert

Name	Typ
PushbackDcTimeValue	SJsonValue

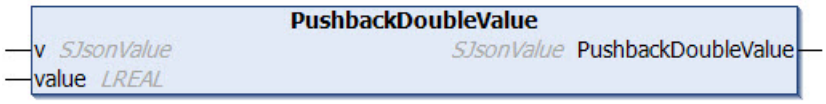
Eingänge

Name	Typ
v	sJsonValue
value	DCTIME

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonArray := fbJson.FindMember(jsonDoc, 'array');
jsonValue := fbJson.PushbackDcTimeValue(jsonArray, dcTime);
```

5.2.1.1.80 PushbackDoubleValue



Diese Methode hängt einen Wert vom Datentyp Double an das Ende eines Arrays an.

Syntax

```
METHOD PushbackDoubleValue : SJsonValue
VAR_INPUT
  v     : SJsonValue;
  value : LREAL;
END_VAR
```

Rückgabewert

Name	Typ
PushbackDoubleValue	SJsonValue

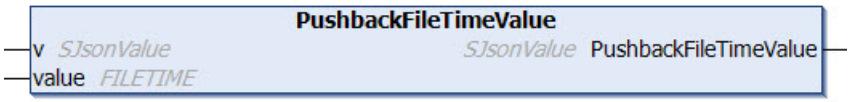
Eingänge

Name	Typ
v	sJsonValue
value	LREAL

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonArray := fbJson.FindMember(jsonDoc, 'array');
jsonValue := fbJson.PushbackDoubleValue(jsonArray, 42.42);
```

5.2.1.1.81 PushbackFileTimeValue



Diese Methode hängt einen Wert vom Datentyp FILETIME an das Ende eines Arrays an.

Syntax

```
METHOD PushbackFileTimeValue : SJsonValue
VAR_INPUT
  v     : SJsonValue;
  value : FILETIME;
END_VAR
```

Rückgabewert

Name	Typ
PushbackFileTimeValue	SJsonValue

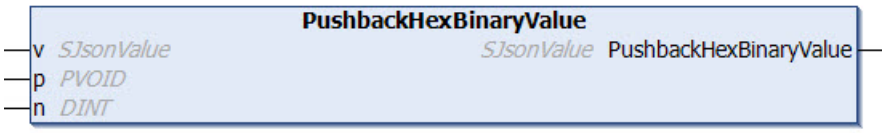
Eingänge

Name	Typ
v	sJsonValue
value	FILETIME

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonArray := fbJson.FindMember(jsonDoc, 'array');
jsonValue := fbJson.PushbackFileTimeValue(jsonArray, fileTime);
```

5.2.1.1.82 PushbackHexBinaryValue



Diese Methode hängt einen HexBinary-kodierten Wert an das Ende eines Arrays an. Die Kodierung in das HexBinary-Format wird durch die Methode durchgeführt. Als Quelle kann z. B. eine Datenstruktur verwendet werden.

Syntax

```
METHOD PushbackHexBinaryValue : SJsonValue
VAR_INPUT
  v : SJsonValue;
  p : PVOID;
  n : DINT;
END_VAR
```

Rückgabewert

Name	Typ
PushbackHexBinaryValue	SJsonValue

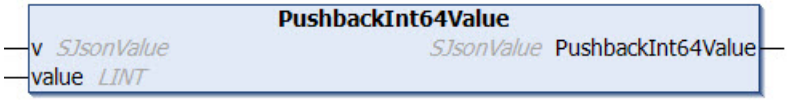
Eingänge

Name	Typ
v	sJsonValue
p	PVOID
n	DINT

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonArray := fbJson.FindMember(jsonDoc, 'array');
jsonValue := fbJson.PushbackHexBinaryValue(jsonArray, ADR(stStruct), SIZEOF(stStruct));
```

5.2.1.1.83 PushbackInt64Value



Diese Methode hängt einen Wert vom Datentyp Int64 an das Ende eines Arrays an.

Syntax

```
METHOD PushbackInt64Value : SJsonValue
VAR_INPUT
  v     : SJsonValue;
  value : LINT;
END_VAR
```

Rückgabewert

Name	Typ
PushbackInt64Value	SJsonValue

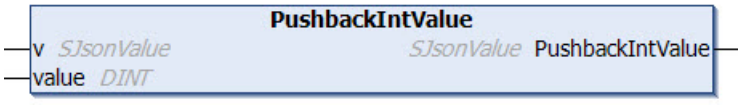
Eingänge

Name	Typ
v	sJsonValue
value	LINT

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonArray := fbJson.FindMember(jsonDoc, 'array');
jsonValue := fbJson.PushbackInt64Value(jsonArray, 42);
```

5.2.1.1.84 PushbackIntValue



Diese Methode hängt einen Wert vom Datentyp INT an das Ende eines Arrays an.

Syntax

```
METHOD PushbackIntValue : SJsonValue
VAR_INPUT
  v     : SJsonValue;
  value : DINT;
END_VAR
```

Rückgabewert

Name	Typ
PushbackIntValue	SJsonValue

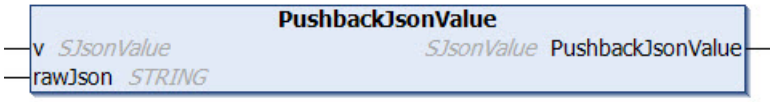
Eingänge

Name	Typ
v	sJsonValue
value	DINT

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonArray := fbJson.FindMember(jsonDoc, 'array');
jsonValue := fbJson.PushbackIntValue(jsonArray, 42);
```

5.2.1.1.85 PushbackJsonValue



Diese Methode fügt ein als STRING in der SPS vorhandenes JSON-Dokument zum Ende eines Arrays hinzu.

Syntax

```
METHOD PushbackJsonValue : SJsonValue
VAR_INPUT
  v       : SJsonValue;
END_VAR
VAR_IN_OUT CONSTANT
  rawJson : STRING;
END_VAR
```

Rückgabewert

Name	Typ
PushbackJsonValue	SJsonValue

Eingänge

Name	Typ
v	SJsonValue

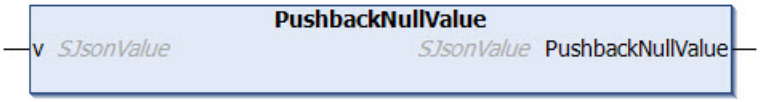
/ Ein-/Ausgänge

Name	Typ
rawJson	STRING

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonArray := fbJson.FindMember(jsonDoc, 'array');
jsonValue := fbJson.PushbackJsonValue(jsonArray, sJson);
```

5.2.1.1.86 PushbackNullValue



Diese Methode hängt einen NULL-Wert an das Ende eines Arrays an.

Syntax

```
METHOD PushbackNullValue : SJsonValue
VAR_INPUT
  v : SJsonValue;
END_VAR
```

Rückgabewert

Name	Typ
PushbackNullValue	SJsonValue

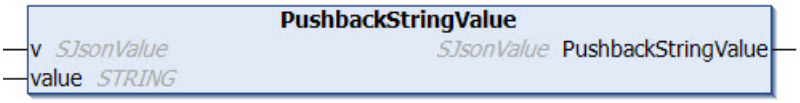
Eingänge

Name	Typ
v	SJsonValue

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonArray := fbJson.FindMember(jsonDoc, 'array');
jsonValue := fbJson.PushbackNullValue(jsonArray);
```

5.2.1.1.87 PushbackStringValue



Diese Methode hängt einen Wert vom Datentyp DCTIME an das Ende eines Arrays an.

Syntax

```
METHOD PushbackStringValue : SJsonValue
VAR_INPUT
  v     : SJsonValue;
END_VAR
VAR_IN_OUT CONSTANT
  value : STRING;
END_VAR
```

Rückgabewert

Name	Typ
PushbackStringValue	SJsonValue

Eingänge

Name	Typ
v	SJsonValue

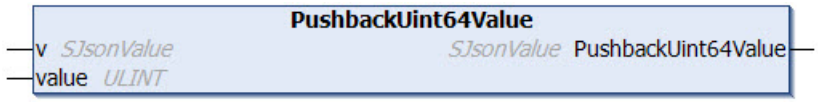
/ Ein-/Ausgänge

Name	Typ
value	STRING

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonArray := fbJson.FindMember(jsonDoc, 'array');
jsonValue := fbJson.PushbackStringValue(jsonArray, sString);
```

5.2.1.1.88 PushbackUint64Value



Diese Methode hängt einen Wert vom Datentyp UInt64 an das Ende eines Arrays an.

Syntax

```
METHOD PushbackUint64Value : SJsonValue
VAR_INPUT
  v     : SJsonValue;
  value : ULINT;
END_VAR
```

Rückgabewert

Name	Typ
PushbackUint64Value	SJsonValue

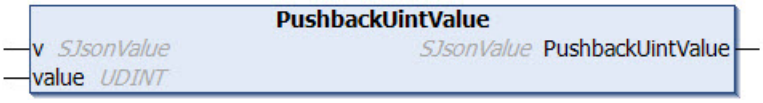
Eingänge

Name	Typ
v	SJsonValue
value	ULINT

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonArray := fbJson.FindMember(jsonDoc, 'array');
jsonValue := fbJson.PushbackUint64Value(jsonArray, 42);
```

5.2.1.1.89 PushbackUintValue



Diese Methode hängt einen Wert vom Datentyp UInt an das Ende eines Arrays an.

Syntax

```
METHOD PushbackUintValue : SJsonValue
VAR_INPUT
  v     : SJsonValue;
  value : UDINT;
END_VAR
```

Rückgabewert

Name	Typ
PushbackUintValue	SJsonValue

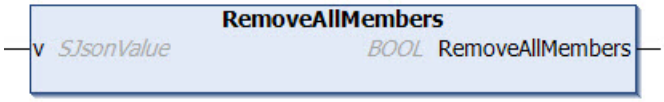
Eingänge

Name	Typ
v	SJsonValue
value	UDINT

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonArray := fbJson.FindMember(jsonDoc, 'array');
jsonValue := fbJson.PushbackUintValue(jsonArray, 42);
```

5.2.1.1.90 RemoveAllMembers



Diese Methode entfernt alle Kindelemente von einem gegebenen Property.

Syntax

```
METHOD RemoveAllMembers : BOOL
VAR_INPUT
  v : SJsonValue;
END_VAR
```

Rückgabewert

Name	Typ
RemoveAllMembers	BOOL

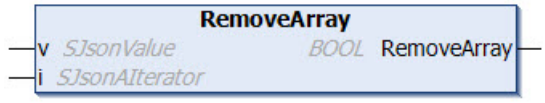
Eingänge

Name	Typ
v	SJsonValue

Beispielaufruf:

```
jsonDoc  := fbJson.ParseDocument(sExistingJsonDocument);
jsonProp := fbJson.FindMember(jsonDoc, 'property');
bRemoved := fbJson.RemoveAllMembers(jsonProp);
```

5.2.1.1.91 RemoveArray



Diese Methode löscht den Wert des aktuellen Array-Iterators.

Syntax

```
METHOD RemoveArray : BOOL
VAR_INPUT
  v : SJsonValue;
  i : SJsonAIterator;
END_VAR
```

Rückgabewert

Name	Typ
RemoveArray	BOOL

Eingänge

Name	Typ
v	SJsonValue
i	SJsonAIterator

Beispielaufruf:

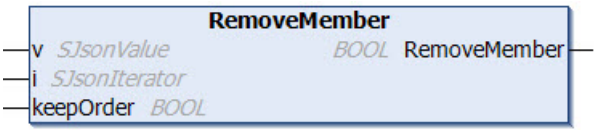
Gegeben sei das folgende JSON-Dokument, das in den DOM-Speicher geladen wird:

```
sMessage := '{"serialNumber":"123","batteryVoltage":"1547mV","clickType":"SINGLE", "array":
["Hello",2,3]}';
```

Die erste Array-Position soll gelöscht werden. Zunächst wird das JSON-Dokument iterativ nach dem Property „array“ durchsucht, anschließend wird das erste Element des Arrays durch Aufruf der Methode RemoveArray()entfernt.

```
jsonDoc               := fbJson.ParseDocument(sMessage);
jsonIterator          := fbJson.MemberBegin(jsonDoc);
jsonIteratorEnd       := fbJson.MemberEnd(jsonDoc);
WHILE jsonIterator <> jsonIteratorEnd DO
  sName               := fbJson.GetMemberName(jsonIterator);
  jsonValue           := fbJson.GetMemberValue(jsonIterator);
  IF sName = 'array' THEN
    jsonArrayIterator := fbJson.ArrayBegin(jsonValue);
    fbJson.RemoveArray(jsonValue, jsonArrayIterator);
  END_IF
  jsonIterator        := fbJson.NextMember(jsonIterator);
END_WHILE
```

5.2.1.1.92 RemoveMember



Diese Methode löscht das Property an dem aktuellen Iterator.

Syntax

```
METHOD RemoveMember : BOOL
VAR_INPUT
  v : SJsonValue;
  i : SJsonIterator;
  keepOrder : BOOL;
END_VAR
```

Rückgabewert

Name	Typ
RemoveMember	BOOL

Eingänge

Name	Typ
v	SJsonValue
i	SJsonIterator
keepOrder	BOOL

Beispielaufruf:

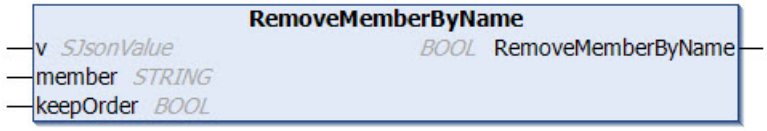
Gegeben sei das folgende JSON Dokument, das in den DOM-Speicher geladen wird:

```
sMessage := '{"serialNumber":"123","batteryVoltage":"1547mV","clickType":"SINGLE", "array":
["Hello",2,3]}';
```

Das Property „array“ soll gelöscht werden. Zunächst wird das JSON-Dokument iterativ nach dem Property „array“ durchsucht, anschließend wird es entfernt.

```
jsonDoc         := fbJson.ParseDocument(sMessage);
jsonIterator    := fbJson.MemberBegin(jsonDoc);
jsonIteratorEnd := fbJson.MemberEnd(jsonDoc);
WHILE jsonIterator <> jsonIteratorEnd DO
  sName         := fbJson.GetMemberName(jsonIterator);
  IF sName = 'array' THEN
    fbJson.RemoveMember(jsonDoc, jsonIterator);
  END_IF
  jsonIterator  := fbJson.NextMember(jsonIterator);
END_WHILE
```

5.2.1.1.93 RemoveMemberByName



Diese Methode entfernt ein Kindelement von einem gegebenen Property. Das Element wird hierbei über dessen Namen referenziert.

Syntax

```
METHOD RemoveMemberByName : BOOL
VAR_INPUT
  v         : SJsonValue;
  keepOrder : BOOL;
END_VAR
VAR_IN_OUT CONSTANT
    member  : STRING;
END_VAR
```

Rückgabewert

Name	Typ
RemoveMemberByName	BOOL

Eingänge

Name	Typ
v	SJsonValue
keepOrder	BOOL

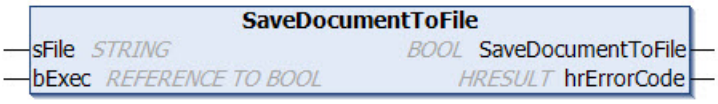
/ Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonProp  := fbJson.FindMember(jsonDoc, 'property');
jsonValue := fbJson.RemoveMemberByName(jsonProp, 'ChildName');
```

5.2.1.1.94 SaveDocumentToFile



Diese Methode speichert ein JSON-Dokument in einer Datei.

Eine steigende Flanke am Eingangsparameter bExec triggert hierbei den Speichervorgang. Der asynchrone Prozess ist beendet, sobald die Referenz bExec von der Methode aus zurück auf FALSE gesetzt wird. Mit Beendigung des Prozesses zeigt der Rückgabewert der Methode für einen Aufruf lang an, ob das Speichern der Datei erfolgreich (TRUE) oder fehlgeschlagen (FALSE) war.

Syntax

```
METHOD SaveDocumentToFile : BOOL
VAR_IN_OUT CONSTANT
    sFile    : STRING;
END_VAR
VAR_INPUT
    bExec    : REFERENCE TO BOOL;
END_VAR
VAR_OUTPUT
  hrErrorCode: HRESULT;
END_VAR
```

Rückgabewert

Name	Typ
SaveDocumentToFile	BOOL

Eingänge

Name	Typ
bExec	REFERENCE TO BOOL

Ein-/Ausgänge

Name	Typ
sFile	STRING

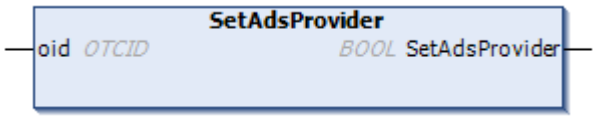
Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

```
IF bSave THEN
    bSaved := fbJson.SaveDocumentToFile(sFile, bSave);
END_IF
```

5.2.1.1.95 SetAdsProvider



Diese Methode setzt den Kontext, in dem die File Access-Zugriffe beim Laden und Speichern der JSON-Dokumente abgearbeitet werden. Wenn kein ADS-Provider angegeben wird, wird die aktuelle Task genutzt.

Syntax

```
METHOD SetAdsProvider : BOOL
VAR_INPUT
  oid : OTCID;
END_VAR
```

Rückgabewert

Name	Typ	Beschreibung
SetAdsProvider	BOOL	Die Methode gibt TRUE zurück, wenn das Setzen des ADS-Providers erfolgreich war.

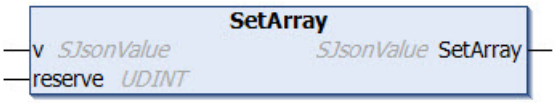
Eingänge

Name	Typ	Beschreibung
oid	OTCID	Object ID der Task.

Beispielaufruf:

```
bTest   := fbJson.SetAdsProvider(02010030);
```

5.2.1.1.96 SetArray



Diese Methode setzt den Value eines Properties auf den Typ „Array“. Neue Werte können nun mit den Pushback-Methoden zum Array hinzugefügt werden.

Syntax

```
METHOD SetArray : SJsonValue
VAR_INPUT
  v : SJsonValue;
END_VAR
```

Rückgabewert

Name	Typ
SetArray	SJsonValue

Eingänge

Name	Typ
v	SJsonValue

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonProp  := fbJson.FindMember(jsonDoc, 'property');
jsonValue := fbJson.SetArray(jsonProp);
```

5.2.1.1.97 SetBase64

Diese Methode setzt den Value eines Properties auf einen Base64-kodierten Wert. Als Quelle kann z. B. eine Datenstruktur verwendet werden. Die Kodierung in das Base64-Format erfolgt innerhalb der Methode.

Syntax

```
METHOD SetBase64 : SJsonValue
VAR_INPUT
  v : SJsonValue;
```

```
  p : PVOID;
  n : DINT;
END_VAR
```

Rückgabewert

Name	Typ
SetBase64	SJsonValue

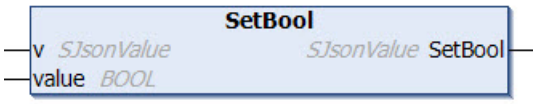
Eingänge

Name	Typ
v	SJsonValue
p	PVOID
n	DINT

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonProp  := fbJson.FindMember(jsonDoc, 'property');
jsonValue := fbJson.SetBase64(jsonProp, ADR(stStruct), SIZEOF(stStruct));
```

5.2.1.1.98 SetBool



Diese Methode setzt den Value eines Properties auf einen Wert vom Datentyp BOOL.

Syntax

```
METHOD SetBool : SJsonValue
VAR_INPUT
  v     : SJsonValue;
  value : BOOL;
END_VAR
```

Rückgabewert

Name	Typ
SetBool	SJsonValue

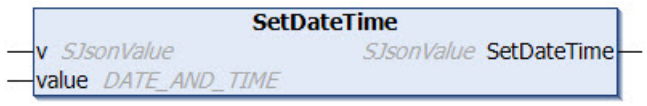
Eingänge

Name	Typ
v	SJsonValue
value	BOOL

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonProp  := fbJson.FindMember(jsonDoc, 'property');
jsonValue := fbJson.SetBool(jsonProp, TRUE);
```

5.2.1.1.99 SetDateTime



Diese Methode setzt den Value eines Properties auf einen Wert vom Datentyp DATE_AND_TIME.

Syntax

```
METHOD SetDateTime : SJsonValue
VAR_INPUT
  v     : SJsonValue;
  value : DATE_AND_TIME;
END_VAR
```

Rückgabewert

Name	Typ
SetDateTime	SJsonValue

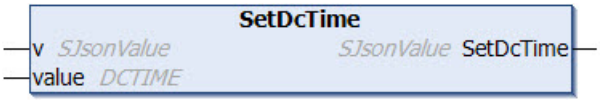
Eingänge

Name	Typ
v	SJsonValue
value	DATE_AND_TIME

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonProp  := fbJson.FindMember(jsonDoc, 'property');
jsonValue := fbJson.SetDateTime(jsonProp, dtTime);
```

5.2.1.1.100 SetDcTime



Diese Methode setzt den Value eines Properties auf einen Wert vom Datentyp DCTIME.

Syntax

```
METHOD SetDcTime : SJsonValue
VAR_INPUT
  v     : SJsonValue;
  value : DCTIME;
END_VAR
```

Rückgabewert

Name	Typ
SetDcTime	SJsonValue

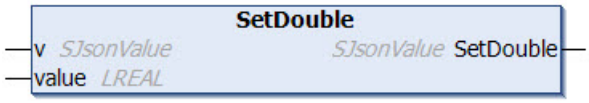
Eingänge

Name	Typ
v	SJsonValue
value	DCTIME

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonProp  := fbJson.FindMember(jsonDoc, 'property');
jsonValue := fbJson.SetDcTime(jsonProp, dcTime);
```

5.2.1.1.101 SetDouble



Diese Methode setzt den Value eines Properties auf einen Wert vom Datentyp Double.

Syntax

```
METHOD SetDouble : SJsonValue
VAR_INPUT
  v     : SJsonValue;
  value : LREAL;
END_VAR
```

Rückgabewert

Name	Typ
SetDouble	SJsonValue

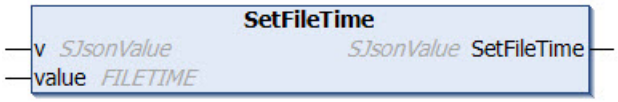
Eingänge

Name	Typ
v	SJsonValue
value	LREAL

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonProp  := fbJson.FindMember(jsonDoc, 'property');
jsonValue := fbJson.SetDouble(jsonProp, 42.42);
```

5.2.1.1.102 SetFileTime



Diese Methode setzt den Value eines Properties auf einen Wert vom Datentyp FILETIME.

Syntax

```
METHOD SetFileTime : SJsonValue
VAR_INPUT
  v     : SJsonValue;
  value : FILETIME;
END_VAR
```

Rückgabewert

Name	Typ
SetFileTime	SJsonValue

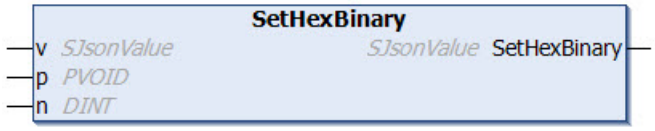
Eingänge

Name	Typ
v	SJsonValue
value	FILETIME

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonProp  := fbJson.FindMember(jsonDoc, 'property');
jsonValue := fbJson.SetFileTime(jsonProp, fileTime);
```

5.2.1.1.103 SetHexBinary



Diese Methode setzt den Value eines Properties auf einen HexBinary-kodierten Wert. Als Quelle kann z. B. eine Datenstruktur verwendet werden. Die Kodierung in das HexBinary-Format erfolgt innerhalb der Methode.

Syntax

```
METHOD SetHexBinary : SJsonValue
VAR_INPUT
  v : SJsonValue;
  p : PVOID;
  n : DINT;
END_VAR
```

Rückgabewert

Name	Typ
SetHexBinary	SJsonValue

Eingänge

Name	Typ
v	SJsonValue
p	PVOID
n	DINT

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonProp  := fbJson.FindMember(jsonDoc, 'property');
jsonValue := fbJson.SetHexBinary(jsonProp, ADR(stStruct), SIZEOF(stStruct));
```

5.2.1.1.104 SetInt

Diese Methode setzt den Value eines Properties auf einen Wert vom Datentyp INT.

Syntax

```
METHOD SetInt : SJsonValue
VAR_INPUT
  v     : SJsonValue;
  value : DINT;
END_VAR
```

Rückgabewert

Name	Typ
SetInt	SJsonValue

Eingänge

Name	Typ
v	SJsonValue
value	DINT

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonProp  := fbJson.FindMember(jsonDoc, 'property');
jsonValue := fbJson.SetInt(jsonProp, 42);
```

5.2.1.1.105 SetInt64

Diese Methode setzt den Value eines Properties auf einen Wert vom Datentyp Int64.

Syntax

```
METHOD SetInt64 : SJsonValue
VAR_INPUT
  v     : SJsonValue;
  value : LINT;
END_VAR
```

Rückgabewert

Name	Typ
SetInt64	SJsonValue

Eingänge

Name	Typ
v	SJsonValue
value	LINT

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonProp  := fbJson.FindMember(jsonDoc, 'property');
jsonValue := fbJson.SetInt64(jsonProp, 42);
```

5.2.1.1.106 SetJson

Diese Methode fügt in den Value eines Properties ein weiteres JSON-Dokument ein.

Syntax

```
METHOD SetJson : SJsonValue
VAR_INPUT
  v       : SJsonValue;
END_VAR
VAR_IN_OUT CONSTANT
  rawJson : STRING;
END_VAR
```

Rückgabewert

Name	Typ
SetJson	SJsonValue

Eingänge

Name	Typ
v	SJsonValue

/ Ein-/Ausgänge

Name	Typ
rawJson	STRING

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonProp  := fbJson.FindMember(jsonDoc, 'property');
jsonValue := fbJson.SetJson(jsonProp, sJson);
```

5.2.1.1.107 SetMaxDecimalPlaces

Diese Funktion legt die Anzahl an Nachkommastellen fest, nach denen eine Gleitkommazahl abgeschnitten wird.

Syntax

```
METHOD SetMaxDecimalPlaces
VAR_INPUT
  dp : DINT;
END_VAR
```

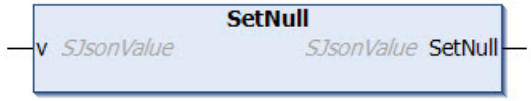
Eingänge

Name	Typ
dp	DINT

Beispielaufruf:

```
nDec := fbJson.SetMaxDecimalPlaces();
```

5.2.1.1.108 SetNull



Diese Methode setzt den Value eines Properties auf den Wert NULL.

Syntax

```
METHOD SetNull : SJsonValue
VAR_INPUT
  v : SJsonValue;
END_VAR
```

Rückgabewert

Name	Typ
SetNull	SJsonValue

Eingänge

Name	Typ
v	SJsonValue

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonProp  := fbJson.FindMember(jsonDoc, 'property');
jsonValue := fbJson.SetNull(jsonProp);
```

5.2.1.1.109 SetObject

Diese Methode setzt den Value eines Properties auf den Typ „Object“. Dies ermöglicht eine Verschachtelung von JSON-Dokumenten.

Syntax

```
METHOD SetObject : SJsonValue
VAR_INPUT
  v     : SJsonValue;
END_VAR
```

Rückgabewert

Name	Typ
SetObject	SJsonValue

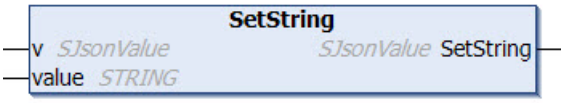
Eingänge

Name	Typ
v	SJsonValue

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonProp  := fbJson.FindMember(jsonDoc, 'property');
jsonValue := fbJson.SetObject(jsonProp);
```

5.2.1.1.110 SetString



Diese Methode setzt den Value eines Properties auf einen Wert vom Datentyp STRING.

Syntax

```
METHOD SetString : SJsonValue
VAR_INPUT
  v     : SJsonValue;
END_VAR
```

```
VAR_IN_OUT CONSTANT
  value : STRING;
END_VAR
```

Rückgabewert

Name	Typ
SetString	SJsonValue

Eingänge

Name	Typ
v	SJsonValue

/ Ein-/Ausgänge

Name	Typ
value	STRING

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonProp  := fbJson.FindMember(jsonDoc, 'property');
jsonValue := fbJson.SetString(jsonProp, 'Hello World');
```

5.2.1.1.111 SetUint

Diese Methode setzt den Value eines Properties auf einen Wert vom Datentyp UInt.

Syntax

```
METHOD SetUint: SJsonValue
VAR_INPUT
  v     : SJsonValue;
  value : UDINT;
END_VAR
```

Rückgabewert

Name	Typ
SetUint	SJsonValue

Eingänge

Name	Typ
v	SJsonValue
value	UDINT

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonProp  := fbJson.FindMember(jsonDoc, 'property');
jsonValue := fbJson.SetUint(jsonProp, 42);
```

5.2.1.1.112 SetUint64

Diese Methode setzt den Value eines Properties auf einen Wert vom Datentyp UInt64.

Syntax

```
METHOD SetUint64 : SJsonValue
VAR_INPUT
  v     : SJsonValue;
  value : ULINT;
END_VAR
```

Rückgabewert

Name	Typ
SetUint64	SJsonValue

Eingänge

Name	Typ
v	SJsonValue
value	ULINT

Beispielaufruf:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonProp  := fbJson.FindMember(jsonDoc, 'property');
jsonValue := fbJson.SetUint64(jsonProp, 42);
```

5.2.1.1.113 Swap

Diese Methode tauscht den Inhalt von zwei Pointern.

Syntax

```
METHOD Swap : BOOL
VAR_INPUT
  v     : SJsonValue;
  w     : SJsonValue;
END_VAR;
END_VAR
```

Rückgabewert

Name	Typ
Swap	BOOL

Eingänge

Name	Typ
v	SJsonValue
w	SJsonValue

5.2.1.2 FB_JsonDynDomParser

Dieser Funktionsblock ist von demselben internen Funktionsbaustein abgeleitet wie der FB_JsonDomParser [▶ 119] und bietet somit dasselbe Interface.

Die beiden abgeleiteten Funktionsblöcke unterscheiden sich nur in ihrer internen Speicherverwaltung. Der FB_JsonDynDomParser ist optimiert für JSON-Dokumente, an denen viele Änderungen vorgenommen werden und das JSON-Dokument nicht zwischendurch freigegeben wird. Er gibt den allokierten Speicher nach Ausführung einer Aktion, z.B. bei den Methoden SetObject() oder SetJson(), wieder frei. Durch diese Flexibilität ergibt sich ein größerer Overhead, sodass der FB_JsonDomParser [▶ 119] eine bessere Performance ermöglicht.

Strings im UTF-8-Format

Die hier verwendeten Variablen vom Typ STRING nutzen das UTF-8-Format. Diese STRING-Formatierung ist üblich bei IoT/MQTT-Kommunikation sowie JSON-Dokumenten.

Um Sonderzeichen und Texte verschiedenster Sprachen empfangen zu können, wird der Zeichensatz in den Bibliotheken Tc3_IotBase und Tc3_JsonXml nicht auf den typischen Zeichensatz vom Datentyp STRING beschränkt. Stattdessen wird der Unicode-Zeichensatz als UTF-8-Format in Verbindung mit dem Datentyp STRING verwendet.

Bei Verwendung des ASCII-Zeichensatzes besteht kein Unterschied zwischen der typischen Formatierung in einem STRING und der UTF-8-Formatierung eines STRING.

Weitere Informationen zum UTF-8-STRING-Format sowie vorhandenen Anzeige- und Konvertierungsmöglichkeiten finden Sie in der Dokumentation der SPS-Bibliothek Tc2_Utilities.

Syntax

```
FUNCTION_BLOCK FB_JsonDynDomParser
VAR_OUTPUT
  initStatus     : HRESULT;
END_VAR
```

Ausgänge

Name	Typ
initStatus	HRESULT

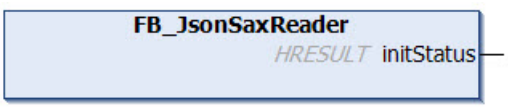
Methoden

Alle Methoden finden Sie in FB_JsonDomParser [▶ 119].

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4024.7	x86, x64, ARM	Tc3_JsonXml 3.3.8.0

5.2.1.3 FB_JsonSaxReader



Syntax

```
FUNCTION_BLOCK FB_JsonSaxReader
VAR_OUTPUT
  initStatus     : HRESULT;
END_VAR
```

Ausgänge

Name	Typ
initStatus	HRESULT

Methoden

Name	Beschreibung
DecodeBase64 [▶ 190]	Konvertiert einen Base64-formatierten String in Binärdaten.
DecodeDateTime [▶ 191]	Erzeugt aus einem genormten ISO8601-Zeitformat (z. B. YYYY-MM-DDThh:mm:ss) eine SPS-Variable vom Datentyp DATE_AND_TIME bzw. DT.
DecodeDcTime [▶ 192]	Erzeugt aus einem genormten ISO8601 Zeitformat (z. B. YYYY-MM-DDThh:mm:ss) eine SPS-Variable vom Datentyp DCTIME.
DecodeFileTime [▶ 192]	Erzeugt aus einem genormten ISO8601-Zeitformat (z. B. YYYY-MM-DDThh:mm:ss) eine SPS-Variable vom Datentyp FILETIME.
DecodeHexBinary [▶ 193]	Konvertiert einen String, der Hexadezimalwerte enthält, in Binärdaten um.
IsBase64 [▶ 194]	Prüft, ob der übergebene String dem Base64-Format entspricht.
IsHexBinary [▶ 194]	Prüft, ob der übergebene String aus Hexadezimalwerten besteht.
IsISO8601TimeFormat [▶ 195]	Prüft, ob der übergebene String dem genormten ISO8601-Zeitformat entspricht.
Parse [▶ 195]	Startet den SAX-Reader-Parsing-Vorgang.
ParseValues [▶ 196]	Startet den SAX-Reader-Parsing-Vorgang.

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4022	x86, x64, ARM	Tc3_JsonXml

5.2.1.3.1 DecodeBase64

Diese Methode konvertiert einen Base64-formatierten String in Binärdaten. Wenn die Konvertierung erfolgreich war, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

```
METHOD DecodeBase64 : BOOL
VAR_INPUT
  sBase64     : STRING;
  pBytes      : POINTER TO BYTE;
  nBytes      : REFERENCE TO DINT;
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

Rückgabewert

Name	Typ
DecodeBase64	BOOL

Eingänge

Name	Typ
sBase64	STRING
pBytes	POINTER TO BYTE
nBytes	REFERENCE TO DINT

Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

```
bSuccess := fbJson.DecodeBase64('SGVsbG8gVHdpbkNBVA==', ADR(byteArray), byteArraySize);
```

5.2.1.3.2 DecodeDateTime

Diese Methode ermöglicht es, aus einem genormten ISO8601-Zeitformat (z. B. YYYY-MM-DDThh:mm:ss) eine SPS-Variable vom Datentyp DATE_AND_TIME bzw. DT zu erzeugen. DT entspricht hierbei der Anzahl an Sekunden ab dem Datum 01.01.1970. Wenn die Konvertierung erfolgreich war, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

```
METHOD DecodeDateTime : BOOL
VAR_IN_OUT CONSTANT
  sDT          : STRING;
END_VAR
VAR_OUTPUT
  nDT          : DATE_AND_TIME;
  hrErrorCode  : HRESULT;
END_VAR
```

Rückgabewert

Name	Typ
DecodeDateTime	BOOL

Ein-/Ausgänge

Name	Typ
sDT	STRING

Ausgänge

Name	Typ
nDT	DATE_AND_TIME
hrErrorCode	HRESULT

Beispielaufruf:

```
bSuccess := fbJson.DecodeDateTime('2017-08-09T06:54:00', nDT => dateTime);
```

5.2.1.3.3 DecodeDcTime

Diese Methode ermöglicht es, aus einem genormten ISO8601 Zeitformat (z. B. YYYY-MM-DDThh:mm:ss) eine SPS-Variable vom Datentyp DCTIME zu erzeugen. DCTIME entspricht der Anzahl an Nanosekunden ab dem Datum 01.01.2000. Wenn die Konvertierung erfolgreich war, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

```
METHOD DecodeDcTime : BOOL
VAR_IN_OUT CONSTANT
  sDC          : STRING;
END_VAR
VAR_OUTPUT
  nDC          : DCTIME;
  hrErrorCode : HRESULT;
END_VAR
```

Rückgabewert

Name	Typ
DecodeDcTime	BOOL

/ Ein-/Ausgänge

Name	Typ
sDC	STRING

Ausgänge

Name	Typ
nDC	DCTIME
hrErrorCode	HRESULT

Beispielaufruf:

```
bSuccess := fbJson.DecodeDcTime('2017-08-09T06:54:00', nDc => dcTime);
```

5.2.1.3.4 DecodeFileTime

Diese Methode ermöglicht es, aus einem genormten ISO8601-Zeitformat (z. B. YYYY-MM-DDThh:mm:ss) eine SPS-Variable vom Datentyp FILETIME zu erzeugen. FILETIME entspricht der Anzahl an Nanosekunden ab dem Datum 01.01.1601 – gemessen in 100 Nanosekunden. Wenn die Konvertierung erfolgreich war, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

```
METHOD DecodeDateTime : BOOL
VAR_IN_OUT CONSTANT
  sFT          : STRING;
END_VAR
VAR_OUTPUT
```

```
  nFT         : FILETIME;
  hrErrorCode : HRESULT;
END_VAR
```

Rückgabewert

Name	Typ
DecodeDateTime	BOOL

/ Ein-/Ausgänge

Name	Typ
sFT	STRING

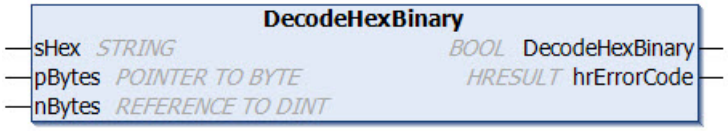
Ausgänge

Name	Typ
nFT	FILETIME
hrErrorCode	HRESULT

Beispielaufruf:

```
bSuccess := fbJson.DecodeFileTime('2017-08-09T06:54:00', nFT => fileTime);
```

5.2.1.3.5 DecodeHexBinary



Diese Methode konvertiert einen String, der Hexadezimalwerte enthält, in Binärdaten um. Wenn die Konvertierung erfolgreich war, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

```
METHOD DecodeHexBinary : BOOL
VAR_IN_OUT CONSTANT
  sHex        : STRING;
END_VAR
VAR_INPUT
  pBytes      : POINTER TO BYTE;
  nBytes      : REFERENCE TO DINT;
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

Rückgabewert

Name	Typ
DecodeHexBinary	BOOL

Eingänge

Name	Typ
pBytes	POINTER TO BYTE
nBytes	REFERENCE TO DINT

/ Ein-/Ausgänge

Name	Typ
sHex	STRING

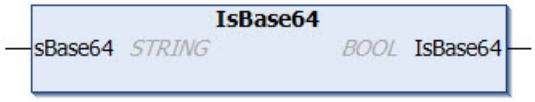
Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

```
bSuccess := fbJson.DecodeHexBinary('ABCEF93A', ADR(byteArray), byteArraySize);
```

5.2.1.3.6 IsBase64



Diese Methode prüft, ob der übergebene String dem Base64-Format entspricht. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

```
METHOD IsBase64 : BOOL
VAR_IN_OUT CONSTANT
  sBase64 : STRING;
END_VAR
```

Rückgabewert

Name	Typ
IsBase64	BOOL

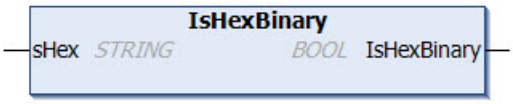
/ Ein-/Ausgänge

Name	Typ
sBase64	STRING

Beispielaufruf:

```
bIsBase64 := fbJson.IsBase64('SGVsbG8gVHdpbkNBVA==');
```

5.2.1.3.7 IsHexBinary



Diese Methode prüft, ob der übergebene String aus Hexadezimalwerten besteht. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

```
METHOD IsHexBinary : BOOL
VAR_IN_OUT CONSTANT
  sHex : STRING;
END_VAR
```

Rückgabewert

Name	Typ
IsHexBinary	BOOL

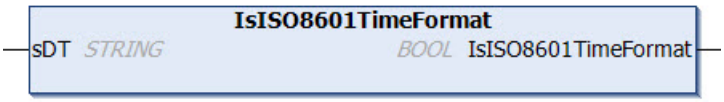
/ Ein-/Ausgänge

Name	Typ
sHex	STRING

Beispielaufruf:

```
bSuccess := fbJson.IsHexBinary('ABCEF93A');
```

5.2.1.3.8 IsISO8601TimeFormat



Diese Methode prüft, ob der übergebene String dem genormten ISO8601-Zeitformat entspricht. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

```
METHOD IsISO8601TimeFormat : BOOL
VAR_IN_OUT CONSTANT
  sDT : STRING;
END_VAR
```

Rückgabewert

Name	Typ
IsISO8601TimeFormat	BOOL

/ Ein-/Ausgänge

Name	Typ
sDT	STRING

Beispielaufruf:

```
bSuccess := fbJson.IsISO8601TimeFormat('2017-08-09T06:54:00');
```

5.2.1.3.9 Parse

Durch diese Methode wird der SAX-Reader-Parsing-Vorgang gestartet. Als Eingangsparameter werden das zu parsende JSON-Objekt und eine Referenz auf einen Funktionsbaustein übergeben, der vom Interface ITcJsonSaxHandler abgeleitet wurde. Dieser Funktionsbaustein wird dann für die Callback-Methoden des SAX Readers verwendet.

Syntax

```
METHOD Parse : BOOL
VAR_IN_OUT CONSTANT
  sJson       : STRING;
```

```
END_VAR
VAR_INPUT
  ipHdl       : ITcJsonSaxHandler;
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

Rückgabewert

Name	Typ
Parse	BOOL

Eingänge

Name	Typ
ipHdl	ITcJsonSaxHandler

/ Ein-/Ausgänge

Name	Typ
sJson	STRING

Ausgänge

Name	Typ
hrErrorCode	HRESULT

5.2.1.3.10 ParseValues

Durch diese Methode wird der SAX-Reader-Parsing-Vorgang gestartet. Als Eingangsparameter werden das zu parsende JSON-Objekt und eine Referenz auf einen Funktionsbaustein übergeben, der vom Interface ITcJsonSaxValues abgeleitet wurde. Dieser Funktionsbaustein wird dann für die Callback-Methoden des SAX Readers verwendet. Die Besonderheit bei dieser Methode ist, dass bei den Callback-Methoden ausschließlich Values berücksichtigt werden, d. h. es gibt keine OnKey()- oder OnStartObject()-Callbacks.

Syntax

```
METHOD ParseValues : BOOL
VAR_IN_OUT CONSTANT
  sJson       : STRING;
END_VAR
VAR_INPUT
  ipHdl       : ITcJsonSaxValues;
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

Rückgabewert

Name	Typ
ParseValues	BOOL

Eingänge

Name	Typ
ipHdl	ITcJsonSaxValues

/ Ein-/Ausgänge

Name	Typ
sJson	STRING

Ausgänge

Name	Typ
hrErrorCode	HRESULT

5.2.1.4 FB_JsonSaxWriter

Strings im UTF-8-Format

Die hier verwendeten Variablen vom Typ STRING nutzen das UTF-8-Format. Diese STRING-Formatierung ist üblich bei IoT/MQTT-Kommunikation sowie JSON-Dokumenten.

Um Sonderzeichen und Texte verschiedenster Sprachen empfangen zu können, wird der Zeichensatz in den Bibliotheken Tc3_IotBase und Tc3_JsonXml nicht auf den typischen Zeichensatz vom Datentyp STRING beschränkt. Stattdessen wird der Unicode-Zeichensatz als UTF-8-Format in Verbindung mit dem Datentyp STRING verwendet.

Bei Verwendung des ASCII-Zeichensatzes besteht kein Unterschied zwischen der typischen Formatierung in einem STRING und der UTF-8-Formatierung eines STRING.

Weitere Informationen zum UTF-8-STRING-Format sowie vorhandenen Anzeige- und Konvertierungsmöglichkeiten finden Sie in der Dokumentation der SPS-Bibliothek Tc2_Utilities.

Syntax

```
FUNCTION_BLOCK FB_JsonSaxWriter
VAR_OUTPUT
  initStatus     : HRESULT;
END_VAR
```

Ausgänge

Name	Typ
initStatus	HRESULT

Methoden

Name	Beschreibung
AddBase64 [▶ 200]	Fügt einen Wert vom Datentyp Base64 zu einem Property hinzu.
AddBool [▶ 200]	Fügt einen Wert vom Datentyp BOOL zu einem Property hinzu.
AddDateTime [▶ 201]	Fügt einen Wert vom Datentyp DATE_AND_TIME zu einem Property hinzu.
AddDcTime [▶ 201]	Fügt einen Wert vom Datentyp DCTIME zu einem Property hinzu.
AddDint [▶ 201]	Fügt einen Wert vom Datentyp DINT zu einem Property hinzu.
AddFileTime [▶ 202]	Fügt einen Wert vom Datentyp FILETIME zu einem Property hinzu.
AddHexBinary [▶ 202]	Fügt einen HexBinary-Wert zu einem Property hinzu.
AddKey [▶ 203]	Fügt einen neuen Property-Key an der aktuellen Position des SAX Writers hinzu.
AddKeyBool [▶ 203]	Legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp BOOL an.
AddKeyDateTime [▶ 204]	Legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp DATE_AND_TIME an.
AddKeyDcTime [▶ 204]	Legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp DCTIME an.
AddKeyFileTime [▶ 205]	Legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp FILETIME an.
AddKeyLreal [▶ 205]	Legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp LREAL an.
AddKeyNull [▶ 206]	Legt einen neuen Property-Key an und initialisiert dessen Wert mit null.
AddKeyNumber [▶ 206]	Legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp DINT an.
AddKeyString [▶ 207]	Legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp STRING an.
AddLint [▶ 207]	Fügt einen Wert vom Datentyp LINT zu einem Property hinzu.
AddLreal [▶ 208]	Fügt einen Wert vom Datentyp LREAL zu einem Property hinzu.
AddNull [▶ 208]	Fügt den Wert null zu einem Property hinzu.
AddRawArray [▶ 208]	Fügt ein gültiges JSON-Array zu einem gegebenen Property als Value hinzu.
AddRawObject [▶ 209]	Fügt ein gültiges JSON-Objekt zu einem gegebenen Property als Value hinzu.
AddReal [▶ 209]	Fügt einen Wert vom Datentyp REAL zu einem Property hinzu.
AddString [▶ 210]	Fügt einen Wert vom Datentyp STRING zu einem Property hinzu.
AddUdint [▶ 210]	Fügt einen Wert vom Datentyp UDINT zu einem Property hinzu.
AddUlint [▶ 211]	Fügt einen Wert vom Datentyp ULINT zu einem Property hinzu.
CopyDocument [▶ 211]	Kopiert den Inhalt des aktuell mit dem SAX Writer erstellten JSON-Objekts in eine Zielvariable vom Datentyp STRING.
EndArray [▶ 212]	Erzeugt den Abschluss eines angefangenen JSON-Arrays und fügt sie an der aktuellen Position des SAX Writers ein.
EndObject [▶ 212]	Erzeugt den Abschluss eines angefangenen JSON-Objekts und fügt sie an der aktuellen Position des SAX Writers ein.
GetDocument [▶ 213]	Liefert den Inhalt des aktuell mit dem SAX Writer erstellten JSON-Objekts und gibt diesen als Datentyp STRING(255) zurück.
GetDocumentLength [▶ 213]	Liefert die Länge des aktuell mit dem SAX Writer erstellten JSON-Objekts und gibt diese als Datentyp UDINT zurück.
GetMaxDecimalPlaces [▶ 214]	
IsComplete	
ResetDocument [▶ 214]	Setzt das aktuell mit dem SAX Writer erstellte JSON-Objekt zurück.
SetMaxDecimalPlaces [▶ 214]	

Name	Beschreibung
StartArray [▶ 215]	Erzeugt den Anfang eines neuen JSON-Arrays und fügt sie an der aktuellen Position des SAX Writers ein.
StartObject [▶ 215]	Erzeugt den Beginn eines neuen JSON-Objekts und fügt sie an der aktuellen Position des SAX Writers ein.

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4022	x86, x64, ARM	Tc3_JsonXml

5.2.1.4.1 AddBase64

Diese Methode fügt einen Wert vom Datentyp Base64 zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 203] ein entsprechendes Property erstellt.

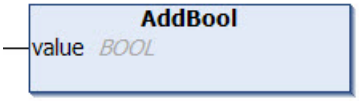
Syntax

```
METHOD AddBase64
VAR_INPUT
  pBytes : Pointer TO BYTE;
  nBytes : DINT;
END_VAR
```

Eingänge

Name	Typ
pBytes	POINTER TO BYTE
nBytes	DINT

5.2.1.4.2 AddBool



Diese Methode fügt einen Wert vom Datentyp BOOL zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 203] ein entsprechendes Property erstellt.

Syntax

```
METHOD AddBool
VAR_INPUT
  value : BOOL;
END_VAR
```

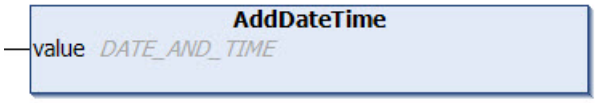
Eingänge

Name	Typ
value	BOOL

Beispielaufruf:

```
fbJson.AddKey('bSwitch');
fbJson.AddBool(TRUE);
```

5.2.1.4.3 AddDateTime



Diese Methode fügt einen Wert vom Datentyp DATE_AND_TIME zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 203] ein entsprechendes Property erstellt.

Syntax

```
METHOD AddDateTime
VAR_INPUT
  value : DATE_AND_TIME;
END_VAR
```

Eingänge

Name	Typ
value	DATE_AND_TIME

Beispielaufruf:

```
fbJson.AddKey('Timestamp');
fbJson.AddDateTime(dtTime); // dtTime is of type DATE_AND_TIME
```

5.2.1.4.4 AddDcTime

Diese Methode fügt einen Wert vom Datentyp DCTIME zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 203] ein entsprechendes Property erstellt.

Syntax

```
METHOD AddDcTime
VAR_INPUT
  value : DCTIME;
END_VAR
```

Eingänge

Name	Typ
value	DCTIME

Beispielaufruf:

```
fbJson.AddKey('Timestamp');
fbJson.AddDcTime(dcTime); // dcTime is of type DCTIME
```

5.2.1.4.5 AddDint

Diese Methode fügt einen Wert vom Datentyp DINT zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 203] ein entsprechendes Property erstellt.

Syntax

```
METHOD AddDint
VAR_INPUT
  value : DINT;
END_VAR
```

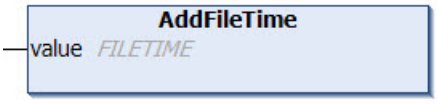
Eingänge

Name	Typ
value	DINT

Beispielaufruf:

```
fbJson.AddKey('nNumber');
fbJson.AddDint(42);
```

5.2.1.4.6 AddFileTime



Diese Methode fügt einen Wert vom Datentyp FILETIME zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 203] ein entsprechendes Property erstellt.

Syntax

```
METHOD AddFileTime
VAR_INPUT
  value : FILETIME;
END_VAR
```

Eingänge

Name	Typ
value	FILETIME

Beispielaufruf:

```
fbJson.AddKey('Timestamp');
fbJson.AddFileTime(ftTime); // ftTime is of type FILETIME
```

5.2.1.4.7 AddHexBinary

Diese Methode fügt einen HexBinary-Wert zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 203] ein entsprechendes Property erstellt.

Syntax

```
METHOD AddHexBinary
VAR_INPUT
  pBytes : POINTER TO BYTE;
  nBytes : DINT;
END_VAR
```

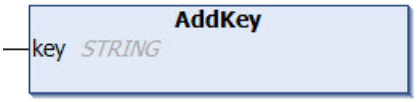
Eingänge

Name	Typ
pBytes	POINTER TO BYTE
nBytes	DINT

Beispielaufruf:

```
fbJson.AddKey('HexBinary');
fbJson.AddHexBinary(ADR(byteHexBin), SIZEOF(byteHexBin));
```

5.2.1.4.8 AddKey



Diese Methode fügt einen neuen Property-Key an der aktuellen Position des SAX Writers hinzu. Üblicherweise wird anschließend der Value des neuen Properties gesetzt. Dies kann zum Beispiel über eine der folgenden Methoden erfolgen: AddBase64 [▶ 200], AddBool [▶ 200], AddDateTime [▶ 201], AddDcTime [▶ 201], AddDint [▶ 201], AddFileTime [▶ 202], AddHexBinary [▶ 202], AddLint [▶ 207], AddLreal [▶ 208], AddNull [▶ 208], AddRawArray [▶ 208], AddRawObject [▶ 209], AddReal [▶ 209], AddString [▶ 210], AddUdint [▶ 210], AddUlint [▶ 211].

Syntax

```
METHOD AddKey
VAR_IN_OUT CONSTANT
  key : STRING;
END_VAR
```

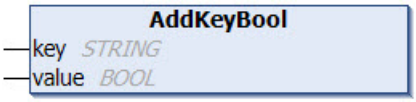
/ Ein-/Ausgänge

Name	Typ
key	STRING

Beispielaufruf:

```
fbJson.AddKey('PropertyName');
```

5.2.1.4.9 AddKeyBool



Diese Methode legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp BOOL an.

Syntax

```
METHOD AddKeyBool
VAR_IN_OUT CONSTANT
  key   : STRING;
END_VAR
VAR_INPUT
  value : BOOL;
END_VAR
```

Eingänge

Name	Typ
value	BOOL

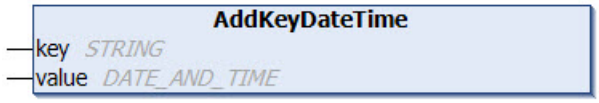
/ Ein-/Ausgänge

Name	Typ
key	STRING

Beispielaufruf:

```
fbJson.AddKeyBool('bSwitch', TRUE);
```

5.2.1.4.10 AddKeyDateTime



Diese Methode legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp DATE_AND_TIME an.

Syntax

```
METHOD AddKeyDateTime
VAR_IN_OUT CONSTANT
  key   : STRING;
END_VAR
VAR_INPUT
  value : DATE_AND_TIME;
END_VAR
```

Eingänge

Name	Typ
value	DATE_AND_TIME

/ Ein-/Ausgänge

Name	Typ
key	STRING

Beispielaufruf:

```
fbJson.AddKeyDateTime('Timestamp', dtTime);
```

5.2.1.4.11 AddKeyDcTime

Diese Methode legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp DCTIME an.

Syntax

```
METHOD AddKeyDcTime
VAR_IN_OUT CONSTANT
  key   : STRING;
END_VAR
VAR_INPUT
  value : DCTIME;
END_VAR
```

Eingänge

Name	Typ
value	DCTIME

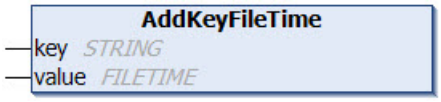
/ Ein-/Ausgänge

Name	Typ
key	STRING

Beispielaufruf:

```
fbJson.AddKeyDcTime('Timestamp', dcTime);
```

5.2.1.4.12 AddKeyFileTime



Diese Methode legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp FILETIME an.

Syntax

```
METHOD AddKeyFileTime
VAR_IN_OUT CONSTANT
  key   : STRING;
END_VAR
VAR_INPUT
  value : FILETIME;
END_VAR
```

Eingänge

Name	Typ
value	FILETIME

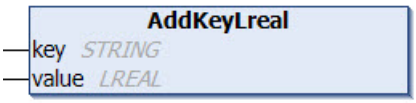
/ Ein-/Ausgänge

Name	Typ
key	STRING

Beispielaufruf:

```
fbJson.AddKeyFileTime('Timestamp', ftTime);
```

5.2.1.4.13 AddKeyLreal



Diese Methode legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp LREAL an.

Syntax

```
METHOD AddKeyLreal
VAR_IN_OUT CONSTANT
  key   : STRING;
END_VAR
VAR_INPUT
  value : LREAL;
END_VAR
```

Eingänge

Name	Typ
value	LREAL

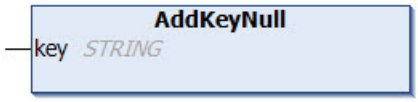
/ Ein-/Ausgänge

Name	Typ
key	STRING

Beispielaufruf:

```
fbJson.AddKeyLreal('PropertyName', 42.42);
```

5.2.1.4.14 AddKeyNull



Diese Methode legt einen neuen Property-Key an und initialisiert dessen Wert mit „null“.

Syntax

```
METHOD AddKeyNull
VAR_IN_OUT CONSTANT
  key : STRING;
END_VAR
```

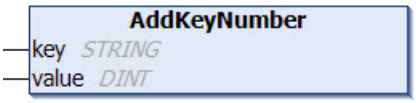
/ Ein-/Ausgänge

Name	Typ
key	STRING

Beispielaufruf:

```
fbJson.AddKeyNull('PropertyName');
```

5.2.1.4.15 AddKeyNumber



Diese Methode legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp DINT an.

Syntax

```
METHOD AddKeyNumber
VAR_IN_OUT CONSTANT
  key   : STRING;
END_VAR
VAR_INPUT
  value : DINT;
END_VAR
```

Eingänge

Name	Typ
value	DINT

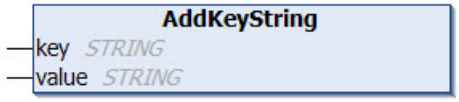
/ Ein-/Ausgänge

Name	Typ
key	STRING

Beispielaufruf:

```
fbJson.AddKeyNumber('PropertyName', 42);
```

5.2.1.4.16 AddKeyString



Diese Methode legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp STRING an.

Syntax

```
METHOD AddKeyString
VAR_IN_OUT CONSTANT
  key   : STRING;
  value : STRING;
END_VAR
```

/ Ein-/Ausgänge

Name	Typ
key	STRING
value	STRING

Beispielaufruf:

```
fbJson.AddKeyString('PropertyName', 'Hello World');
```

5.2.1.4.17 AddLint

Diese Methode fügt einen Wert vom Datentyp LINT zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 203] ein entsprechendes Property erstellt.

Syntax

```
METHOD AddLint
VAR_INPUT
  value : LINT;
END_VAR
```

Eingänge

Name	Typ
value	LINT

Beispielaufruf:

```
fbJson.AddKey('PropertyName');
fbJson.AddLint(42);
```

5.2.1.4.18 AddLreal

Diese Methode fügt einen Wert vom Datentyp LREAL zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 203] ein entsprechendes Property erstellt.

Syntax

```
METHOD AddLreal
VAR_INPUT
  value : LREAL;
END_VAR
```

Eingänge

Name	Typ
value	LREAL

Beispielaufruf:

```
fbJson.AddKey('PropertyName');
fbJson.AddLreal(42.42);
```

5.2.1.4.19 AddNull

Diese Methode fügt den Wert „null“ zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 203] ein entsprechendes Property erstellt.

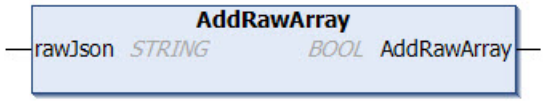
Syntax

```
METHOD AddNull
```

Beispielaufruf:

```
fbJson.AddKey('PropertyName');
fbJson.AddNull();
```

5.2.1.4.20 AddRawArray



Diese Methode fügt ein gültiges JSON-Array zu einem gegebenen Property als Value hinzu. Das hinzuzufügende Array muss ein gültiges JSON-Format aufweisen und darf nur dann hinzugefügt werden, wenn der SAX Writer sich an einer entsprechend gültigen Position befindet, also z. B. direkt nach einem vorhergehenden AddKey() [▶ 203], StartArray() [▶ 215] oder als erster Aufruf nach einem ResetDocument() [▶ 214].

Syntax

```
METHOD AddRawArray
VAR_IN_OUT CONSTANT
  rawJson : STRING;
END_VAR
```

Rückgabewert

Name	Typ
AddRawArray	BOOL

Ein-/Ausgänge

Name	Typ
rawJson	STRING

Beispielaufruf:

```
fbJson.AddKey('PropertyName');
fbJson.AddRawArray('[1, 2, {"x":42, "y":42}, 4');
```

5.2.1.4.21 AddRawObject

Diese Methode fügt ein gültiges JSON-Objekt zu einem gegebenen Property als Value hinzu. Das hinzuzufügende Objekt muss ein gültiges JSON-Format aufweisen und darf nur dann hinzugefügt werden, wenn der SAX Writer sich an einer entsprechend gültigen Position befindet, also z. B. direkt nach einem vorhergehenden AddKey() [▶ 203], StartArray() [▶ 215] oder als erster Aufruf nach einem ResetDocument() [▶ 214].

Syntax

```
METHOD AddRawObject
VAR_IN_OUT CONSTANT
  rawJson : STRING;
END_VAR
```

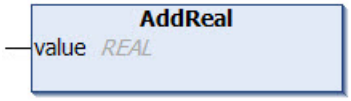
Ein-/Ausgänge

Name	Typ
rawJson	STRING

Beispielaufruf:

```
fbJson.AddKey('PropertyName');
fbJson.AddRawObject('{"x":42, "y":42}');
```

5.2.1.4.22 AddReal



Diese Methode fügt einen Wert vom Datentyp REAL zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 203] ein entsprechendes Property erstellt.

Syntax

```
METHOD AddReal
VAR_INPUT
  value : REAL;
END_VAR
```

Eingänge

Name	Typ
value	REAL

Beispielaufruf:

```
fbJson.AddKey('PropertyName');
fbJson.AddReal(42.42);
```

5.2.1.4.23 AddString

Diese Methode fügt einen Wert vom Datentyp STRING zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 203] ein entsprechendes Property erstellt.

Syntax

```
METHOD AddString
VAR_IN_OUT CONSTANT
  value : STRING;
END_VAR
```

Ein-/Ausgänge

Name	Typ
value	STRING

Beispielaufruf:

```
fbJson.AddKey('PropertyName');
fbJson.AddString('Hello World');
```

5.2.1.4.24 AddUdint

Diese Methode fügt einen Wert vom Datentyp UDINT zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 203] ein entsprechendes Property erstellt.

Syntax

```
METHOD AddUdint
VAR_INPUT
  value : UDINT;
END_VAR
```

Eingänge

Name	Typ
value	UDINT

Beispielaufruf:

```
fbJson.AddKey('PropertyName');
fbJson.AddUdint(42);
```

5.2.1.4.25 AddUlint

Diese Methode fügt einen Wert vom Datentyp ULINT zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 203] ein entsprechendes Property erstellt.

Syntax

```
METHOD AddUlint
VAR_INPUT
  value : ULINT;
END_VAR
```

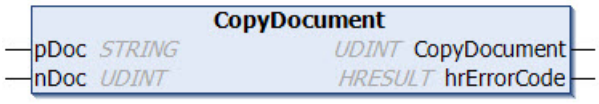
Eingänge

Name	Typ
value	ULINT

Beispielaufruf:

```
fbJson.AddKey('PropertyName');
fbJson.AddUlint(42);
```

5.2.1.4.26 CopyDocument



Diese Methode kopiert den Inhalt des aktuell mit dem SAX Writer erstellten JSON-Objekts in eine Zielvariable vom Datentyp STRING, welche eine beliebige Länge haben kann. Als Rückgabewert liefert die Methode die Länge des Strings (inklusive Nullterminierung). Falls der Zielpuffer zu klein ist, wird dieser durch eine Nullterminierung geleert und als Länge 0 zurückgegeben.

Syntax

```
METHOD CopyDocument : UDINT
VAR_IN_OUT CONSTANT
  pDoc       : STRING;
END_VAR
VAR_INPUT
  nDoc       : UDINT;
END_VAR
VAR_OUTPUT
  hrErrorCode: HRESULT;
END_VAR
```

Rückgabewert

Name	Typ
CopyDocument	UDINT

Eingänge

Name	Typ
nDoc	UDINT

/ Ein-/Ausgänge

Name	Typ
pDoc	STRING

Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

```
fbJson.CopyDocument(sTargetString, SIZEOF(sTargetString));
```

5.2.1.4.27 EndArray

Diese Methode erzeugt den Abschluss eines angefangenen JSON-Arrays („eckige schließende Klammer“) und fügt sie an der aktuellen Position des SAX Writers ein.

Syntax

```
METHOD EndArray : HRESULT
```

Rückgabewert

Name	Typ
EndArray	HRESULT

Beispielaufruf:

```
fbJson.EndArray();
```

5.2.1.4.28 EndObject

Diese Methode erzeugt den Abschluss eines angefangenen JSON-Objekts („geschweifte schließende Klammer“) und fügt sie an der aktuellen Position des SAX Writers ein.

Syntax

```
METHOD EndObject : HRESULT
```

Rückgabewert

Name	Typ
EndObject	HRESULT

Beispielaufruf:

```
fbJson.EndObject();
```

5.2.1.4.29 GetDocument

Diese Methode liefert den Inhalt des aktuell mit dem SAX Writer erstellten JSON-Objekts und gibt diesen als Datentyp STRING(255) zurück.

Die maximale Größe des von der Methode zurückgelieferten Strings ist 255 Zeichen. Bei längeren Zeichenfolgen wird von der Methode ein NULL-String zurückgegeben. In diesem Fall muss die Methode CopyDocument [▶ 211]() verwendet werden.

Syntax

```
METHOD GetDocument : STRING(255)
VAR_OUTPUT
  hrErrorCode: HRESULT;
END_VAR
```

Rückgabewert

Name	Typ
GetDocument	STRING(255)

Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

```
sTargetString := fbJson.GetDocument();
```

5.2.1.4.30 GetDocumentLength

Diese Methode liefert die Länge des aktuell mit dem SAX Writer erstellten JSON-Objekts und gibt diese als Datentyp UDINT zurück.

Syntax

```
METHOD GetDocumentLength : UDINT
VAR_OUTPUT
  hrErrorCode: HRESULT;
END_VAR
```

Rückgabewert

Name	Typ
GetDocumentLength	UDINT

Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

```
nLength := fbJson.GetDocumentLength();
```

5.2.1.4.31 GetMaxDecimalPlaces

Diese Funktion fragt die aktuell eingestellte Anzahl an Nachkommastellen fest, nach denen eine Gleitkommazahl abgeschnitten wird.

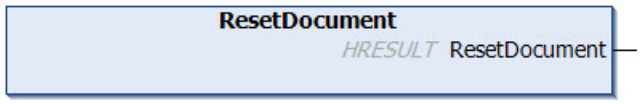
Syntax

```
METHOD GetMaxDecimalPlaces : DINT
```

Rückgabewert

Name	Typ	Beschreibung
GetMaxDecimalPlaces	DINT	Die aktuell eingestellte Anzahl an Stellen, nach der eine Gleitkommazahl abgeschnitten wird.

5.2.1.4.32 ResetDocument



Diese Methode setzt das aktuell mit dem SAX Writer erstellte JSON-Objekt zurück.

Syntax

```
METHOD ResetDocument : HRESULT
```

Rückgabewert

Name	Typ
ResetDocument	HRESULT

Beispielaufruf:

```
fbJson.ResetDocument();
```

5.2.1.4.33 SetMaxDecimalPlaces

Diese Funktion legt die Anzahl an Nachkommastellen fest, nach denen eine Gleitkommazahl abgeschnitten wird.

Syntax

```
METHOD SetMaxDecimalPlaces : HRESULT
VAR_INPUT
    decimalPlaces: DINT;
END_VAR
```

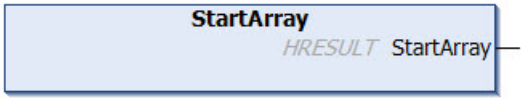
Rückgabewert

Name	Typ
SetMaxDecimalPlaces	HRESULT

Eingänge

Name	Typ	Beschreibung
decimalPlaces	DINT	Die einzustellende Anzahl an Nachkommastellen, nach der eine Gleitkommazahl abgeschnitten wird.

5.2.1.4.34 StartArray



Diese Methode erzeugt den Anfang eines neuen JSON-Arrays („eckige öffnende Klammer“) und fügt sie an der aktuellen Position des SAX Writers ein.

Syntax

```
METHOD StartArray : HRESULT
```

Rückgabewert

Name	Typ
StartArray	HRESULT

Beispielaufruf:

```
fbJson.StartArray();
```

5.2.1.4.35 StartObject

Diese Methode erzeugt den Beginn eines neuen JSON-Objekts („geschweifte öffnende Klammer“) und fügt sie an der aktuellen Position des SAX Writers ein.

Syntax

```
METHOD StartObject : HRESULT
```

Rückgabewert

Name	Typ
StartObject	HRESULT

Beispielaufruf:

```
fbJson.StartObject();
```

5.2.1.5 FB_JsonSaxPrettyWriter

Dieser Funktionsbaustein hat einen Unterschied zum FB_JsonSaxWriter [▶ 197]: Es werden für eine bessere Lesbarkeit eines JSON-Dokuments passende Whitespaces hinzugefügt.

Strings im UTF-8-Format

Die hier verwendeten Variablen vom Typ STRING nutzen das UTF-8-Format. Diese STRING-Formatierung ist üblich bei IoT/MQTT-Kommunikation sowie JSON-Dokumenten.

Um Sonderzeichen und Texte verschiedenster Sprachen empfangen zu können, wird der Zeichensatz in den Bibliotheken Tc3_IotBase und Tc3_JsonXml nicht auf den typischen Zeichensatz vom Datentyp STRING beschränkt. Stattdessen wird der Unicode-Zeichensatz als UTF-8-Format in Verbindung mit dem Datentyp STRING verwendet.

Bei Verwendung des ASCII-Zeichensatzes besteht kein Unterschied zwischen der typischen Formatierung in einem STRING und der UTF-8-Formatierung eines STRING.

Weitere Informationen zum UTF-8-STRING-Format sowie vorhandenen Anzeige- und Konvertierungsmöglichkeiten finden Sie in der Dokumentation der SPS-Bibliothek Tc2_Utilities.

Syntax

```
FUNCTION_BLOCK FB_JsonSaxPrettyWriter
VAR_OUTPUT
  initStatus     : HRESULT;
END_VAR
```

Ausgänge

Name	Typ
initStatus	HRESULT

Methoden

Name	Beschreibung
SetFormatOptions [▶ 231]	Passt die allgemeinen Formatierungsoptionen für eine Instanz des [<=>] FB_JsonSaxPrettyWriter an.
SetIndent [▶ 231]	Passt die Einrückung benutzerdefiniert an.

Alle weiteren Methoden finden Sie beim FB_JsonSaxWriter [▶ 197].

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4022	x86, x64, ARM	Tc3_JsonXml

5.2.1.5.1 AddBase64

Diese Methode fügt einen Wert vom Datentyp Base64 zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 219] ein entsprechendes Property erstellt.

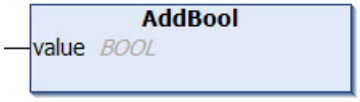
Syntax

```
METHOD AddBase64
VAR_INPUT
  pBytes : Pointer TO BYTE;
  nBytes : DINT;
END_VAR
```

Eingänge

Name	Typ
pBytes	POINTER TO BYTE
nBytes	DINT

5.2.1.5.2 AddBool



Diese Methode fügt einen Wert vom Datentyp BOOL zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 219] ein entsprechendes Property erstellt.

Syntax

```
METHOD AddBool
VAR_INPUT
  value : BOOL;
END_VAR
```

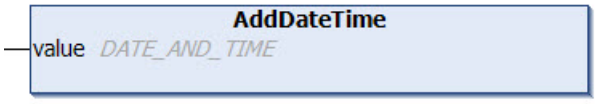
Eingänge

Name	Typ
value	BOOL

Beispielaufruf:

```
fbJson.AddKey('bSwitch');
fbJson.AddBool(TRUE);
```

5.2.1.5.3 AddDateTime



Diese Methode fügt einen Wert vom Datentyp DATE_AND_TIME zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 219] ein entsprechendes Property erstellt.

Syntax

```
METHOD AddDateTime
VAR_INPUT
  value : DATE_AND_TIME;
END_VAR
```

Eingänge

Name	Typ
value	DATE_AND_TIME

Beispielaufruf:

```
fbJson.AddKey('Timestamp');
fbJson.AddDateTime(dtTime); // dtTime is of type DATE_AND_TIME
```

5.2.1.5.4 AddDcTime

Diese Methode fügt einen Wert vom Datentyp DCTIME zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 219] ein entsprechendes Property erstellt.

Syntax

```
METHOD AddDcTime
VAR_INPUT
  value : DCTIME;
END_VAR
```

Eingänge

Name	Typ
value	DCTIME

Beispielaufruf:

```
fbJson.AddKey('Timestamp');
fbJson.AddDcTime(dcTime); // dcTime is of type DCTIME
```

5.2.1.5.5 AddDint

Diese Methode fügt einen Wert vom Datentyp DINT zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 219] ein entsprechendes Property erstellt.

Syntax

```
METHOD AddDint
VAR_INPUT
  value : DINT;
END_VAR
```

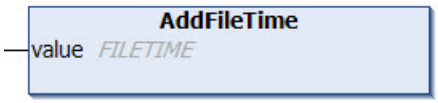
Eingänge

Name	Typ
value	DINT

Beispielaufruf:

```
fbJson.AddKey('nNumber');
fbJson.AddDint(42);
```

5.2.1.5.6 AddFileTime



Diese Methode fügt einen Wert vom Datentyp FILETIME zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 219] ein entsprechendes Property erstellt.

Syntax

```
METHOD AddFileTime
VAR_INPUT
  value : FILETIME;
END_VAR
```

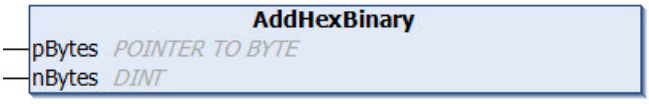
Eingänge

Name	Typ
value	FILETIME

Beispielaufruf:

```
fbJson.AddKey('Timestamp');
fbJson.AddFileTime(ftTime); // ftTime is of type FILETIME
```

5.2.1.5.7 AddHexBinary



Diese Methode fügt einen HexBinary-Wert zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 219] ein entsprechendes Property erstellt.

Syntax

```
METHOD AddHexBinary
VAR_INPUT
  pBytes : POINTER TO BYTE;
  nBytes : DINT;
END_VAR
```

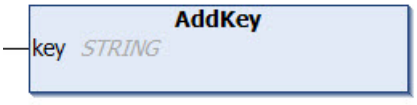
Eingänge

Name	Typ
pBytes	POINTER TO BYTE
nBytes	DINT

Beispielaufruf:

```
fbJson.AddKey('HexBinary');
fbJson.AddHexBinary(ADR(byteHexBin), SIZEOF(byteHexBin));
```

5.2.1.5.8 AddKey



Diese Methode fügt einen neuen Property-Key an der aktuellen Position des SAX Writers hinzu. Üblicherweise wird anschließend der Value des neuen Properties gesetzt. Dies kann zum Beispiel über eine der folgenden Methoden erfolgen: AddBase64 [▶ 216], AddBool [▶ 217], AddDateTime [▶ 217], AddDcTime [▶ 218], AddDint [▶ 218], AddFileTime [▶ 218], AddHexBinary [▶ 219], AddLint [▶ 224], AddLreal [▶ 224], AddNull [▶ 224], AddRawArray [▶ 225], AddRawObject [▶ 225], AddReal [▶ 226], AddString [▶ 226], AddUdint [▶ 227], AddUlint [▶ 227].

Syntax

```
METHOD AddKey
VAR_IN_OUT CONSTANT
  key : STRING;
END_VAR
```

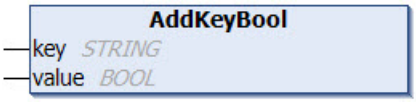
Ein-/Ausgänge

Name	Typ
key	STRING

Beispielaufruf:

```
fbJson.AddKey('PropertyName');
```

5.2.1.5.9 AddKeyBool



Diese Methode legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp BOOL an.

Syntax

```
METHOD AddKeyBool
VAR_IN_OUT CONSTANT
  key   : STRING;
END_VAR
VAR_INPUT
  value : BOOL;
END_VAR
```

Eingänge

Name	Typ
value	BOOL

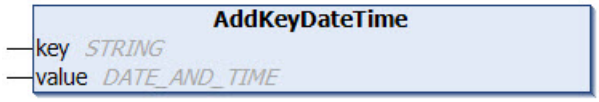
Ein-/Ausgänge

Name	Typ
key	STRING

Beispielaufruf:

```
fbJson.AddKeyBool('bSwitch', TRUE);
```

5.2.1.5.10 AddKeyDateTime



Diese Methode legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp DATE_AND_TIME an.

Syntax

```
METHOD AddKeyDateTime
VAR_IN_OUT CONSTANT
  key   : STRING;
END_VAR
VAR_INPUT
  value : DATE_AND_TIME;
END_VAR
```

Eingänge

Name	Typ
value	DATE_AND_TIME

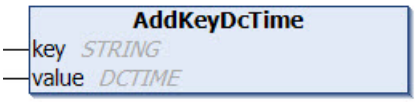
/ Ein-/Ausgänge

Name	Typ
key	STRING

Beispielaufruf:

```
fbJson.AddKeyDateTime('Timestamp', dtTime);
```

5.2.1.5.11 AddKeyDcTime



Diese Methode legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp DCTIME an.

Syntax

```
METHOD AddKeyDcTime
VAR_IN_OUT CONSTANT
  key   : STRING;
END_VAR
VAR_INPUT
  value : DCTIME;
END_VAR
```

Eingänge

Name	Typ
value	DCTIME

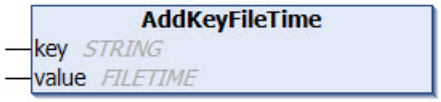
/ Ein-/Ausgänge

Name	Typ
key	STRING

Beispielaufruf:

```
fbJson.AddKeyDcTime('Timestamp', dcTime);
```

5.2.1.5.12 AddKeyFileTime



Diese Methode legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp FILETIME an.

Syntax

```
METHOD AddKeyFileTime
VAR_IN_OUT CONSTANT
  key   : STRING;
END_VAR
VAR_INPUT
  value : FILETIME;
END_VAR
```

Eingänge

Name	Typ
value	FILETIME

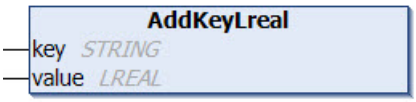
/ Ein-/Ausgänge

Name	Typ
key	STRING

Beispielaufruf:

```
fbJson.AddKeyFileTime('Timestamp', ftTime);
```

5.2.1.5.13 AddKeyLreal



Diese Methode legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp LREAL an.

Syntax

```
METHOD AddKeyLreal
VAR_IN_OUT CONSTANT
  key   : STRING;
END_VAR
VAR_INPUT
  value : LREAL;
END_VAR
```

Eingänge

Name	Typ
value	LREAL

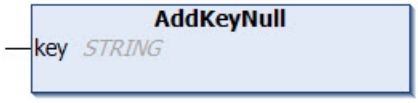
/ Ein-/Ausgänge

Name	Typ
key	STRING

Beispielaufruf:

```
fbJson.AddKeyLreal('PropertyName', 42.42);
```

5.2.1.5.14 AddKeyNull



Diese Methode legt einen neuen Property-Key an und initialisiert dessen Wert mit „null“.

Syntax

```
METHOD AddKeyNull
VAR_IN_OUT CONSTANT
  key : STRING;
END_VAR
```

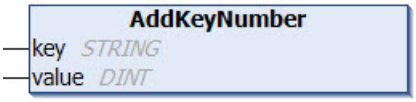
/ Ein-/Ausgänge

Name	Typ
key	STRING

Beispielaufruf:

```
fbJson.AddKeyNull('PropertyName');
```

5.2.1.5.15 AddKeyNumber



Diese Methode legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp DINT an.

Syntax

```
METHOD AddKeyNumber
VAR_IN_OUT CONSTANT
  key   : STRING;
END_VAR
VAR_INPUT
  value : DINT;
END_VAR
```

Eingänge

Name	Typ
value	DINT

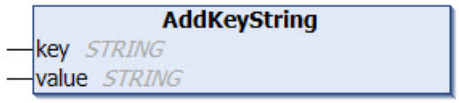
/ Ein-/Ausgänge

Name	Typ
key	STRING

Beispielaufruf:

```
fbJson.AddKeyNumber('PropertyName', 42);
```

5.2.1.5.16 AddKeyString



Diese Methode legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp STRING an.

Syntax

```
METHOD AddKeyString
VAR_IN_OUT CONSTANT
  key   : STRING;
  value : STRING;
END_VAR
```

/ Ein-/Ausgänge

Name	Typ
key	STRING
value	STRING

Beispielaufruf:

```
fbJson.AddKeyString('PropertyName', 'Hello World');
```

5.2.1.5.17 AddLint

Diese Methode fügt einen Wert vom Datentyp LINT zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 219] ein entsprechendes Property erstellt.

Syntax

```
METHOD AddLint
VAR_INPUT
  value : LINT;
END_VAR
```

Eingänge

Name	Typ
value	LINT

Beispielaufruf:

```
fbJson.AddKey('PropertyName');
fbJson.AddLint(42);
```

5.2.1.5.18 AddLreal

Diese Methode fügt einen Wert vom Datentyp LREAL zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 219] ein entsprechendes Property erstellt.

Syntax

```
METHOD AddLreal
VAR_INPUT
  value : LREAL;
END_VAR
```

Eingänge

Name	Typ
value	LREAL

Beispielaufruf:

```
fbJson.AddKey('PropertyName');
fbJson.AddLreal(42.42);
```

5.2.1.5.19 AddNull

Diese Methode fügt den Wert „null“ zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 219] ein entsprechendes Property erstellt.

Syntax

```
METHOD AddNull
```

Beispielaufruf:

```
fbJson.AddKey('PropertyName');
fbJson.AddNull();
```

5.2.1.5.20 AddRawArray

Diese Methode fügt ein gültiges JSON-Array zu einem gegebenen Property als Value hinzu. Das hinzuzufügende Array muss ein gültiges JSON-Format aufweisen und darf nur dann hinzugefügt werden, wenn der SAX Writer sich an einer entsprechend gültigen Position befindet, also z. B. direkt nach einem vorhergehenden AddKey() [▶ 219], StartArray() [▶ 232] oder als erster Aufruf nach einem ResetDocument() [▶ 230].

Syntax

```
METHOD AddRawArray
VAR_IN_OUT CONSTANT
  rawJson : STRING;
END_VAR
```

Rückgabewert

Name	Typ
AddRawArray	BOOL

/ Ein-/Ausgänge

Name	Typ
rawJson	STRING

Beispielaufruf:

```
fbJson.AddKey('PropertyName');
fbJson.AddRawArray('[1, 2, {"x":42, "y":42}, 4');
```

5.2.1.5.21 AddRawObject

Diese Methode fügt ein gültiges JSON-Objekt zu einem gegebenen Property als Value hinzu. Das hinzuzufügende Objekt muss ein gültiges JSON-Format aufweisen und darf nur dann hinzugefügt werden, wenn der SAX Writer sich an einer entsprechend gültigen Position befindet, also z. B. direkt nach einem vorhergehenden AddKey() [▶ 219], StartArray() [▶ 232] oder als erster Aufruf nach einem ResetDocument() [▶ 230].

Syntax

```
METHOD AddRawObject
VAR_IN_OUT CONSTANT
  rawJson : STRING;
END_VAR
```

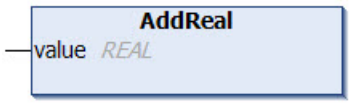
/ Ein-/Ausgänge

Name	Typ
rawJson	STRING

Beispielaufruf:

```
fbJson.AddKey('PropertyName');
fbJson.AddRawObject('{"x":42, "y":42}');
```

5.2.1.5.22 AddReal



Diese Methode fügt einen Wert vom Datentyp REAL zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 219] ein entsprechendes Property erstellt.

Syntax

```
METHOD AddReal
VAR_INPUT
  value : REAL;
END_VAR
```

Eingänge

Name	Typ
value	REAL

Beispielaufruf:

```
fbJson.AddKey('PropertyName');
fbJson.AddReal(42.42);
```

5.2.1.5.23 AddString

Diese Methode fügt einen Wert vom Datentyp STRING zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 219] ein entsprechendes Property erstellt.

Syntax

```
METHOD AddString
VAR_IN_OUT CONSTANT
  value : STRING;
END_VAR
```

/ Ein-/Ausgänge

Name	Typ
value	STRING

Beispielaufruf:

```
fbJson.AddKey('PropertyName');
fbJson.AddString('Hello World');
```

5.2.1.5.24 AddUdint

Diese Methode fügt einen Wert vom Datentyp UDINT zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 219] ein entsprechendes Property erstellt.

Syntax

```
METHOD AddUdint
VAR_INPUT
  value : UDINT;
END_VAR
```

Eingänge

Name	Typ
value	UDINT

Beispielaufruf:

```
fbJson.AddKey('PropertyName');
fbJson.AddUdint(42);
```

5.2.1.5.25 AddUlint

Diese Methode fügt einen Wert vom Datentyp ULINT zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 219] ein entsprechendes Property erstellt.

Syntax

```
METHOD AddUlint
VAR_INPUT
  value : ULINT;
END_VAR
```

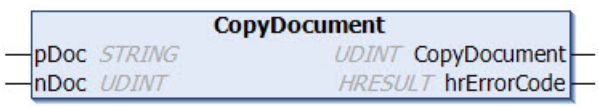
Eingänge

Name	Typ
value	ULINT

Beispielaufruf:

```
fbJson.AddKey('PropertyName');
fbJson.AddUlint(42);
```

5.2.1.5.26 CopyDocument



Diese Methode kopiert den Inhalt des aktuell mit dem SAX Writer erstellten JSON-Objekts in eine Zielvariable vom Datentyp STRING, welche eine beliebige Länge haben kann. Als Rückgabewert liefert die Methode die Länge des Strings (inklusive Nullterminierung). Falls der Zielpuffer zu klein ist, wird dieser durch eine Nullterminierung geleert und als Länge 0 zurückgegeben.

Syntax

```
METHOD CopyDocument : UDINT
VAR_IN_OUT CONSTANT
  pDoc       : STRING;
END_VAR
VAR_INPUT
  nDoc       : UDINT;
END_VAR
VAR_OUTPUT
  hrErrorCode: HRESULT;
END_VAR
```

Rückgabewert

Name	Typ
CopyDocument	UDINT

Eingänge

Name	Typ
nDoc	UDINT

/ Ein-/Ausgänge

Name	Typ
pDoc	STRING

Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

```
fbJson.CopyDocument(sTargetString, SIZEOF(sTargetString));
```

5.2.1.5.27 EndArray

Diese Methode erzeugt den Abschluss eines angefangenen JSON-Arrays („eckige schließende Klammer“) und fügt sie an der aktuellen Position des SAX Writers ein.

Syntax

```
METHOD EndArray : HRESULT
```

Rückgabewert

Name	Typ
EndArray	HRESULT

Beispielaufruf:

```
fbJson.EndArray();
```

5.2.1.5.28 EndObject

Diese Methode erzeugt den Abschluss eines angefangenen JSON-Objekts („geschweifte schließende Klammer“) und fügt sie an der aktuellen Position des SAX Writers ein.

Syntax

```
METHOD EndObject : HRESULT
```

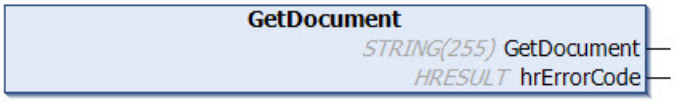
Rückgabewert

Name	Typ
EndObject	HRESULT

Beispielaufruf:

```
fbJson.EndObject();
```

5.2.1.5.29 GetDocument



Diese Methode liefert den Inhalt des aktuell mit dem SAX Writer erstellten JSON-Objekts und gibt diesen als Datentyp STRING(255) zurück.

Die maximale Größe des von der Methode zurückgelieferten Strings ist 255 Zeichen. Bei längeren Zeichenfolgen wird von der Methode ein NULL-String zurückgegeben. In diesem Fall muss die Methode CopyDocument [▶ 227]() verwendet werden.

Syntax

```
METHOD GetDocument : STRING(255)
VAR_OUTPUT
  hrErrorCode: HRESULT;
END_VAR
```

Rückgabewert

Name	Typ
GetDocument	STRING(255)

Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

```
sTargetString := fbJson.GetDocument();
```

5.2.1.5.30 GetDocumentLength

Diese Methode liefert die Länge des aktuell mit dem SAX Writer erstellten JSON-Objekts und gibt diese als Datentyp UDINT zurück.

Syntax

```
METHOD GetDocumentLength : UDINT
VAR_OUTPUT
  hrErrorCode: HRESULT;
END_VAR
```

Rückgabewert

Name	Typ
GetDocumentLength	UDINT

Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

```
nLength := fbJson.GetDocumentLength();
```

5.2.1.5.31 GetMaxDecimalPlaces

Diese Funktion fragt die aktuell eingestellte Anzahl an Nachkommastellen fest, nach denen eine Gleitkommazahl abgeschnitten wird.

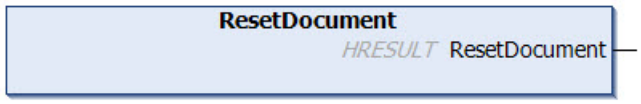
Syntax

```
METHOD GetMaxDecimalPlaces : DINT
```

Rückgabewert

Name	Typ	Beschreibung
GetMaxDecimalPlaces	DINT	Die aktuell eingestellte Anzahl an Stellen, nach der eine Gleitkommazahl abgeschnitten wird.

5.2.1.5.32 ResetDocument



Diese Methode setzt das aktuell mit dem SAX Writer erstellte JSON-Objekt zurück.

Syntax

```
METHOD ResetDocument : HRESULT
```

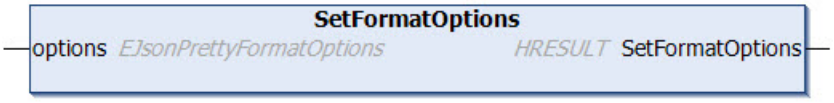
Rückgabewert

Name	Typ
ResetDocument	HRESULT

Beispielaufruf:

```
fbJson.ResetDocument();
```

5.2.1.5.33 SetFormatOptions



Diese Methode wird verwendet, um die allgemeinen Formatierungsoptionen für eine Instanz des FB_JsonSaxPrettyWriter [▶ 215] anzupassen. Zusätzlich zu der Standardkonfiguration gibt es noch die Möglichkeit, dass innerhalb eines Arrays keine Zeilenumbrüche erzeugt werden. Das Array wird dann trotz der eingefügten Einrückungen in einer Zeile ausgegeben.

Syntax

```
METHOD SetFormatOptions : HRESULT
VAR_INPUT
  options               : EJsonPrettyFormatOptions;
END_VAR
TYPE EJsonPrettyFormatOptions:
(
  eFormatDefault        :=0,
  eFormatSingleLineArray:=1
) DINT;
```

Rückgabewert

Name	Typ
SetFormatOptions	HRESULT

Eingänge

Name	Typ
options	EJsonPrettyFormatOptions

Beispielaufruf:

```
fbJson.SetFormatOptions(EJsonPrettyFormatOptions.eFormatSingleLineArray);
```

5.2.1.5.34 SetIndent

Diese Methode wird verwendet, um die Einrückung benutzerdefiniert anzupassen.

Syntax

```
METHOD SetIndent : HRESULT
VAR_INPUT
  indentChar      : SINT;
  indentCharCount : UDINT;
END_VAR
```

Rückgabewert

Name	Typ
SetIndent	HRESULT

Eingänge

Name	Typ
indentChar	SINT
indentCharCount	UDINT

Beispielaufruf:

```
fbJson.SetIndent(1,5);
```

5.2.1.5.35 SetMaxDecimalPlaces

Diese Funktion legt die Anzahl an Nachkommastellen fest, nach denen eine Gleitkommazahl abgeschnitten wird.

Syntax

```
METHOD SetMaxDecimalPlaces : HRESULT
VAR_INPUT
    decimalPlaces: DINT;
END_VAR
```

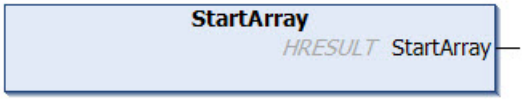
Rückgabewert

Name	Typ
SetMaxDecimalPlaces	HRESULT

Eingänge

Name	Typ	Beschreibung
decimalPlaces	DINT	Die einzustellende Anzahl an Nachkommastellen, nach der eine Gleitkommazahl abgeschnitten wird.

5.2.1.5.36 StartArray



Diese Methode erzeugt den Anfang eines neuen JSON-Arrays („eckige öffnende Klammer“) und fügt sie an der aktuellen Position des SAX Writers ein.

Syntax

```
METHOD StartArray : HRESULT
```

Rückgabewert

Name	Typ
StartArray	HRESULT

Beispielaufruf:

```
fbJson.StartArray();
```

5.2.1.5.37 StartObject

Diese Methode erzeugt den Beginn eines neuen JSON-Objekts („geschweifte öffnende Klammer“) und fügt sie an der aktuellen Position des SAX Writers ein.

Syntax

```
METHOD StartObject : HRESULT
```

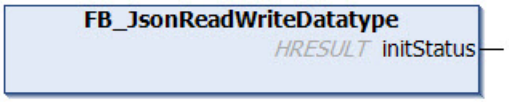
Rückgabewert

Name	Typ
StartObject	HRESULT

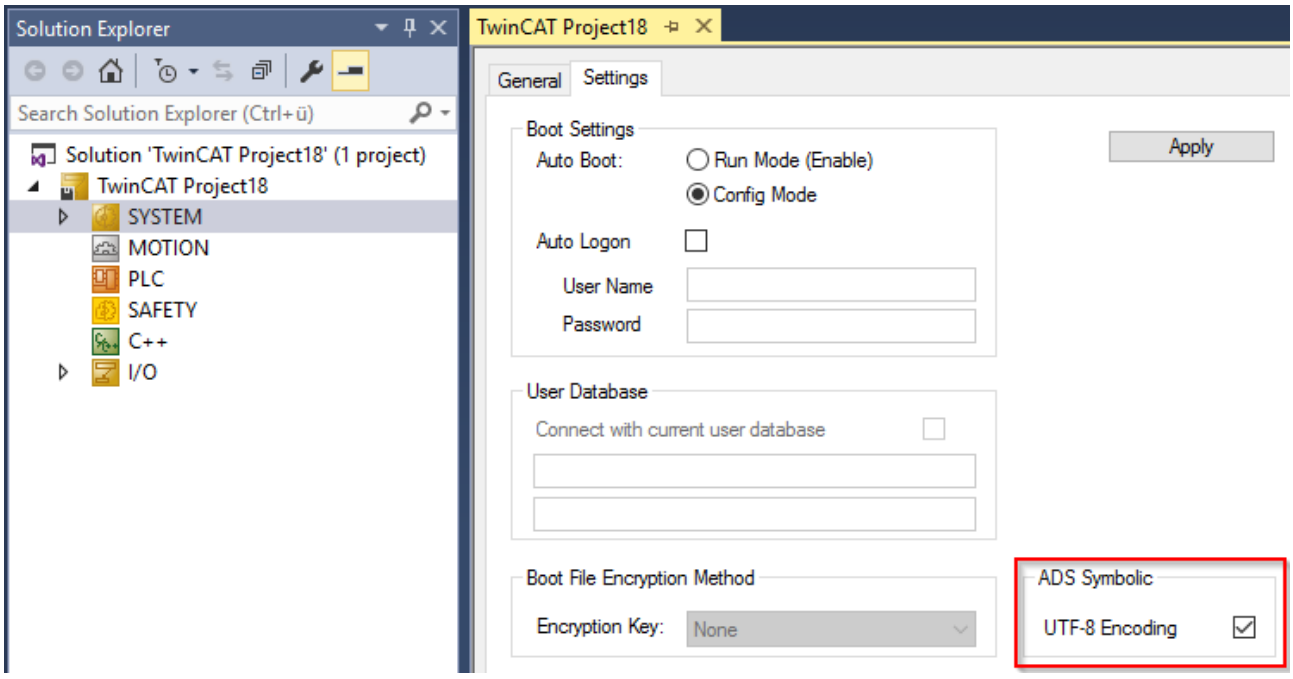
Beispielaufruf:

```
fbJson.StartObject();
```

5.2.1.6 FB_JsonReadWriteDataType



Zur Verwendung von UTF-8-Zeichen, z. B. bei der automatischen Generierung von Metadaten über den Funktionsbaustein FB_JsonReadWriteDataType [▶ 233], muss im TwinCAT-Projekt das Auswahlkästchen zur Unterstützung von UTF-8 in der Symbolik aktiviert sein. Klicken Sie dazu im Projektbaum doppelt auf **SYSTEM**, öffnen Sie die Registerkarte **Settings** und aktivieren Sie das entsprechende Auswahlkästchen.



Strings im UTF-8-Format

Die hier verwendeten Variablen vom Typ STRING nutzen das UTF-8-Format. Diese STRING-Formatierung ist üblich bei IoT/MQTT-Kommunikation sowie JSON-Dokumenten.

Um Sonderzeichen und Texte verschiedenster Sprachen empfangen zu können, wird der Zeichensatz in den Bibliotheken Tc3_IotBase und Tc3_JsonXml nicht auf den typischen Zeichensatz vom Datentyp STRING beschränkt. Stattdessen wird der Unicode-Zeichensatz als UTF-8-Format in Verbindung mit dem Datentyp STRING verwendet.

Bei Verwendung des ASCII-Zeichensatzes besteht kein Unterschied zwischen der typischen Formatierung in einem STRING und der UTF-8-Formatierung eines STRING.

Weitere Informationen zum UTF-8-STRING-Format sowie vorhandenen Anzeige- und Konvertierungsmöglichkeiten finden Sie in der Dokumentation der SPS-Bibliothek Tc2_Utilities.

Ausgänge

Name	Typ
initStatus	HRESULT

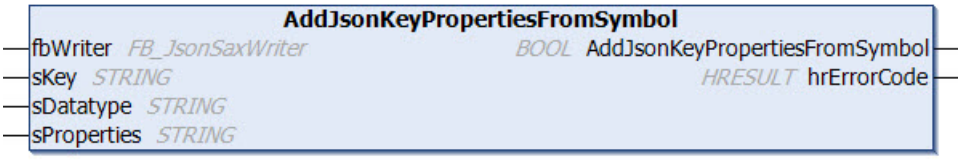
Methoden

Name	Beschreibung
FB_JsonReadWriteDataType	Metadaten werden über SPS-Attribute in die JSON-Repräsentation einer SPS-Datenstruktur auf einem FB_JsonSaxWriter-Objekt hinzugefügt.
AddJsonKeyValueFromSymbol [▶ 236]	Erzeugt die JSON-Repräsentation einer SPS-Datenstruktur auf einem FB_JsonSaxWriter-Objekt.
AddJsonValueFromSymbol [▶ 237]	Erzeugt die JSON-Repräsentation einer SPS-Datenstruktur auf einem FB_JsonSaxWriter-Objekt.
CopyJsonStringFromSymbol [▶ 238]	Erzeugt die JSON-Repräsentanz eines Symbols und kopiert diese in eine Variable vom Datentyp STRING.
CopyJsonStringFromSymbolProperties [▶ 239]	Erzeugt eine entsprechende JSON-Repräsentation von SPS-Attributen an einem Symbol.
CopySymbolNameByAddress [▶ 240]	Liefert den vollständigen (ADS-)Symbolnamen eines übergebenen Symbols.
GetDataTypeNameByAddress [▶ 241]	Liefert den Datentypnamen eines übergebenen Symbols.
GetJsonFromSymbol [▶ 241]	Erzeugt die entsprechende JSON-Repräsentation eines Symbols.
GetJsonStringFromSymbol [▶ 242]	Erzeugt die entsprechende JSON-Repräsentation eines Symbols.
GetJsonStringFromSymbolProperties [▶ 243]	Erzeugt eine entsprechende JSON-Repräsentation von SPS-Attributen an einem Symbol.
GetSizeJsonStringFromSymbol [▶ 244]	Liest die Größe der JSON-Repräsentanz eines Symbols aus.
GetSizeJsonStringFromSymbolProperties [▶ 245]	Liest die Größe der JSON-Repräsentanz von SPS-Attributen an einem Symbol aus.
GetSymbolNameByAddress [▶ 245]	Liefert den vollständigen (ADS-)Symbolnamen eines übergebenen Symbols.
SetSymbolFromJson [▶ 246]	Extrahiert einen String, der eine gültige JSON-Nachricht beinhaltet.

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4022	x86, x64, ARM	Tc3_JsonXml

5.2.1.6.1 AddJsonKeyPropertiesFromSymbol



Mit dieser Methode können Metadaten über SPS-Attribute in die JSON-Repräsentation einer SPS-Datenstruktur auf einem FB_JsonSaxWriter [▶ 197]-Objekt hinzugefügt werden. Als Eingangsparameter erhält die Methode die Instanz des FB_JsonSaxWriter-Funktionsbausteins, den gewünschten Namen des JSON-Properties, das die Metadaten enthalten soll, den Datentypnamen der Struktur und eine String-Variable sProperties, die eine durch einen Querbalken getrennte Liste der zu extrahierenden SPS-Attribute enthält.

Syntax

```
METHOD AddJsonValueFromSymbol : BOOL
VAR_IN_OUT
  fbWriter    : FB_JsonSaxWriter;
END_VAR
VAR_IN_OUT CONSTANT
  sKey        : STRING;
  sDatatype   : STRING;
  sProperties : STRING;
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

Rückgabewert

Name	Typ
AddJsonValueFromSymbol	BOOL

Ein-/Ausgänge

Name	Typ
fbWriter	FB_JsonSaxWriter
sKey	STRING
sDatatype	STRING
sProperties	STRING

Ausgänge

Name	Typ
hrErrorCode	HRESULT

Die SPS-Attribute können in folgender Form an den Strukturelementen spezifiziert werden:

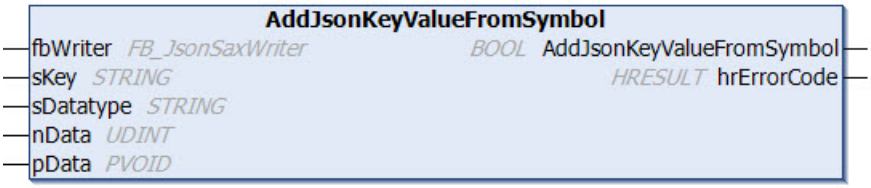
```
{attribute 'Unit'        := 'm/s'}
{attribute 'DisplayName' := 'Speed'}
Sensor1                  : REAL;
```

Ein vollständiges Beispiel zur Verwendung dieser Methode finden Sie im Abschnitt Tc3JsonXmlSampleJsonDataType [▶ 342].

Beispielaufruf:

```
fbJsonSaxWriter.ResetDocument()
fbJsonDataType.AddJsonKeyPropertiesFromSymbol(fbJsonSaxWriter, 'MetaData','ST_Values','Unit|
DisplayName');
```

5.2.1.6.2 AddJsonKeyValueFromSymbol



Diese Methode erzeugt die JSON-Repräsentation einer SPS-Datenstruktur auf einem FB_JsonSaxWriter [▶ 197]-Objekt. Als Eingangsparameter erhält die Methode die Instanz des FB_JsonSaxWriter-Funktionsbausteins, den Datentypnamen der Struktur sowie die Adresse und Größe der Quellstrukturinstanz. Als Resultat beinhaltet die FB_JsonSaxWriter-Instanz eine gültige JSON-Repräsentation der Struktur. Im Unterschied zur Methode AddJsonValueFromSymbol() [▶ 237] werden die Elemente der Quellstruktur hierbei in ein JSON-Unterobjekt verschachtelt, dessen Name über den Eingangs-/Ausgangsparameter sKey spezifiziert werden kann.

Syntax

```
METHOD AddJsonValueFromSymbol : BOOL
VAR_IN_OUT
  fbWriter    : FB_JsonSaxWriter;
END_VAR
VAR_IN_OUT CONSTANT
  sKey        : STRING;
  sDatatype   : STRING;
END_VAR
VAR_INPUT
  nData       : UDINT;
  pData       : PVOID;
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

Rückgabewert

Name	Typ
AddJsonValueFromSymbol	BOOL

Eingänge

Name	Typ
nData	UDINT
pData	PVOID

Ein-/Ausgänge

Name	Typ
fbWriter	FB_JsonSaxWriter
sKey	STRING
sDatatype	STRING

Ausgänge

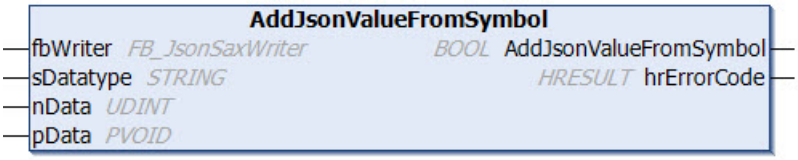
Name	Typ
hrErrorCode	HRESULT

Ein vollständiges Beispiel zur Verwendung dieser Methode finden Sie im Abschnitt Tc3JsonXmlSampleJsonDataType [▶ 342].

Beispielaufruf:

```
fbJsonSaxWriter.ResetDocument()
fbJsonDataType.AddJsonKeyValueFromSymbol(fbJsonSaxWriter, 'Values','ST_Values',SIZEOF(stValues),
ADR(stValues));
```

5.2.1.6.3 AddJsonValueFromSymbol



Diese Methode erzeugt die JSON-Repräsentation einer SPS-Datenstruktur auf einem FB_JsonSaxWriter [▶ 197]-Objekt. Als Eingangsparameter erhält die Methode die Instanz des FB_JsonSaxWriter-Funktionsbausteins, den Datentypnamen der Struktur sowie die Adresse und Größe der Quellstrukturinstanz. Als Resultat beinhaltet die FB_JsonSaxWriter-Instanz eine gültige JSON-Repräsentation der Struktur.

Syntax

```
METHOD AddJsonValueFromSymbol : BOOL
VAR_IN_OUT
  fbWriter : FB_JsonSaxWriter;
END_VAR
VAR_IN_OUT CONSTANT
  sDatatype   : STRING;
END_VAR
VAR_INPUT
  nData       : UDINT;
  pData       : PVOID;
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

Rückgabewert

Name	Typ
AddJsonValueFromSymbol	BOOL

Eingänge

Name	Typ
nData	UDINT
pData	PVOID

/ Ein-/Ausgänge

Name	Typ
fbWriter	FB_JsonSaxWriter
sDatatype	STRING

Ausgänge

Name	Typ
hrErrorCode	HRESULT

Ein vollständiges Beispiel zur Verwendung dieser Methode finden Sie im Abschnitt Tc3JsonXmlSampleJsonDataType [▶ 342].

Beispielaufruf:

```
fbJsonSaxWriter.ResetDocument()
fbJsonDataType.AddJsonValueFromSymbol(fbJsonSaxWriter,'ST_Values',SIZEOF(stValues), ADR(stValues));
```

5.2.1.6.4 CopyJsonStringFromSymbol

Diese Methode erzeugt die JSON-Repräsentanz eines Symbols und kopiert diese in eine Variable vom Datentyp STRING, welche eine beliebige Länge haben kann. Als Rückgabewert liefert die Methode die Länge des Strings (inklusive Nullterminierung). Falls der Zielpuffer zu klein ist, wird dieser durch eine Nullterminierung geleert und als Länge 0 zurückgegeben.

Syntax

```
METHOD CopyJsonStringFromSymbol : UDINT
VAR_INPUT
  nData       : UDINT;
  nDoc        : UDINT;
  pData       : PVOID;
END_VAR
VAR_IN_OUT CONSTANT
  pDoc        : STRING;
  sDatatype   : STRING;
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

Rückgabewert

Name	Typ
CopyJsonStringFromSymbol	UDINT

Eingänge

Name	Typ
nData	UDINT
nDoc	UDINT
pData	PVOID

Ein-/Ausgänge

Name	Typ
pDoc	STRING
sDatatype	STRING

Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

```
nLen :=
fbJsonDataType.CopyJsonStringFromSymbol('ST_Test',SIZEOF(stTest),ADR(stTest),sString,SIZEOF(sString)
);
```

5.2.1.6.5 CopyJsonStringFromSymbolProperties

Diese Methode erzeugt eine entsprechende JSON-Repräsentation von SPS-Attributen an einem Symbol. Im Unterschied zur Methode AddJsonKeyPropertiesFromSymbol [▶ 235] wird das Resultat nicht in eine Instanz vom Funktionsbaustein FB_JsonSaxWriter geschrieben, sondern in eine String-Variable. Als Eingangsparameter erhält die Methode den Datentypnamen des Symbols und eine String-Variable, welche eine durch Querbalken getrennte Liste der zu extrahierenden SPS-Attribute darstellt.

Die Methode kopiert diese JSON-Repräsentation in eine Variable vom Datentyp STRING, welche eine beliebige Länge haben kann. Als Rückgabewert liefert die Methode die Länge des Strings (inklusive Nullterminierung). Falls der Zielpuffer zu klein ist, wird dieser durch eine Nullterminierung geleert und als Länge 0 zurückgegeben.

Syntax

```
METHOD CopyJsonStringFromSymbolProperties : UDINT
VAR_INPUT
  nDoc        : UDINT;
END_VAR
VAR_IN_OUT CONSTANT
  pDoc        : STRING;
  sDatatype   : STRING;
  sProperties : STRING;
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

Rückgabewert

Name	Typ
CopyJsonStringFromSymbolProperties	UDINT

Eingänge

Name	Typ
nDoc	UDINT

/ Ein-/Ausgänge

Name	Typ
pDoc	STRING
sDatatype	STRING
sProperties	STRING

Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

```
nLen := fbJsonDataType.CopyJsonStringFromSymbolProperties('ST_Test','Unit|
DisplayName',sString,SIZEOF(sString));
```

5.2.1.6.6 CopySymbolNameByAddress

Diese Methode liefert den vollständigen (ADS-)Symbolnamen eines übergebenen Symbols. Als Rückgabewert liefert die Methode die Größe des Strings (inklusive Nullterminierung). Falls der Zielpuffer zu klein ist, wird dieser durch eine Nullterminierung geleert und als Länge 0 zurückgegeben.

Syntax

```
METHOD CopySymbolNameByAddress : UDINT
VAR_INPUT
    nData     : UDINT;    // size of symbol
    pData     : PVOID;    // address of symbol
END_VAR
VAR_IN_OUT CONSTANT
    sName     : STRING;    // target string buffer where the symbol name should be copied to
END_VAR
VAR_INPUT
    nName     : UDINT;    // size in bytes of target string buffer
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

Rückgabewert

Name	Typ
CopySymbolNameByAddress	UDINT

Eingänge

Name	Typ
nData	UDINT
pData	PVOID
nName	UDINT

Ein-/Ausgänge

Name	Typ
sName	STRING

Ausgänge

Name	Typ
hrErrorCode	HRESULT

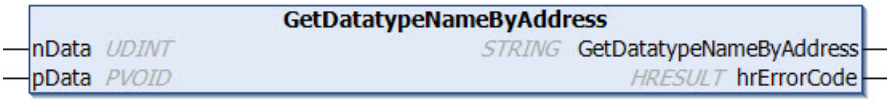
Beispielaufruf:

```
nSymbolSize := fbJsonDataType.CopySymbolNameByAddress(nData:=SIZEOF(stValues), pData:=ADR(stValues),
sName:=sSymbolName, nName:=SIZEOF(sSymbolName));
```

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4024.20	x86, x64, ARM	Tc3_JsonXml 3.3.15.0

5.2.1.6.7 GetDataTypeNameByAddress



Diese Methode liefert den Datentypnamen eines übergebenen Symbols.

Syntax

```
METHOD GetDataTypeNameByAddress : STRING
VAR_INPUT
  nData       : UDINT;
  pData       : PVOID;
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

Rückgabewert

Name	Typ
GetDataTypeNameByAddress	STRING

Eingänge

Name	Typ
nData	UDINT
pData	PVOID

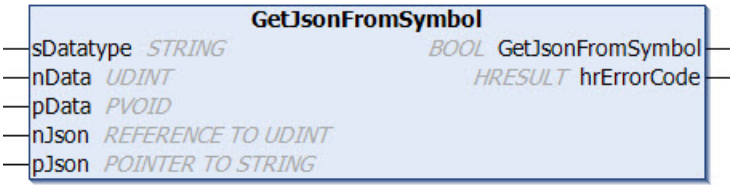
Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

```
sBuffer := fbJsonDataType.GetDataTypeNameByAddress(SIZEOF(stValues),ADR(stValues));
```

5.2.1.6.8 GetJsonFromSymbol



Diese Methode erzeugt die entsprechende JSON-Repräsentation eines Symbols. Im Unterschied zur Methode AddJsonValueFromSymbol() [▶ 237] wird das Resultat nicht in eine Instanz vom Funktionsbaustein FB_JsonSaxWriter geschrieben, sondern in eine String-Variable. Als Eingangsparameter erhält die Methode den Datentypnamen des Symbols sowie die Adresse und Größe des Quellsymbols, z. B. einer Strukturinstanz. Als weitere Eingangsparameter werden die Adresse und Größe des Ziel-Buffers übergeben, der nach dem Aufruf die JSON-Repräsentation des Symbols enthält.

Syntax

```
METHOD GetJsonFromSymbol : BOOL
VAR_IN_OUT CONSTANT
  sDatatype : STRING;
END_VAR
```

```
VAR_INPUT
  nData       : UDINT;
  pData       : PVOID;
  nJson       : REFERENCE TO UDINT;
  pJson       : POINTER TO STRING;
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

Rückgabewert

Name	Typ
GetJsonFromSymbol	BOOL

Eingänge

Name	Typ
nData	UDINT
pData	PVOID
nJson	REFERENCE TO UDINT
pJson	POINTER TO STRING

Ein-/Ausgänge

Name	Typ
sDatatype	STRING

Ausgänge

Name	Typ
hrErrorCode	HRESULT

Eingangsparameter nJson

Der Eingangsparameter nJson enthält beim Aufruf der Methode die Größe des Ziel-Buffers und nach Abschluss des Methodenaufrufs die Größe des tatsächlich geschriebenen JSON-Objekts im Ziel-Buffer.

Beispielaufruf:

```
fbJsonDataType.GetJsonFromSymbol('ST_Values',SIZEOF(stValues), ADR(stValues), nBufferLength,
ADR(sBuffer));
```

5.2.1.6.9 GetJsonStringFromSymbol

Diese Methode erzeugt die entsprechende JSON-Repräsentation eines Symbols. Im Unterschied zur Methode AddJsonValueFromSymbol() [▶ 237] wird das Resultat nicht in eine Instanz vom Funktionsbaustein FB_JsonSaxWriter geschrieben, sondern in eine String-Variable. Als Eingangsparameter erhält die Methode den Datentypnamen des Symbols sowie die Adresse und die Größe des Quellsymbols, z. B. einer Strukturinstanz.

Die maximale Größe des von der Methode zurückgelieferten Strings ist 255 Zeichen. Bei längeren Zeichenfolgen wird von der Methode ein NULL-String zurückgegeben. In diesem Fall muss die Methode CopyJsonStringFromSymbol [▶ 238]() verwendet werden.

Syntax

```
METHOD GetJsonStringFromSymbol : STRING(255)
VAR_IN_OUT CONSTANT
  sDatatype    : STRING;
END_VAR
VAR_INPUT
  nData        : UDINT;
  pData        : PVOID;
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

Rückgabewert

Name	Typ
GetJsonStringFromSymbol	STRING(255)

Eingänge

Name	Typ
nData	UDINT
pData	PVOID

/ Ein-/Ausgänge

Name	Typ
sDatatype	STRING

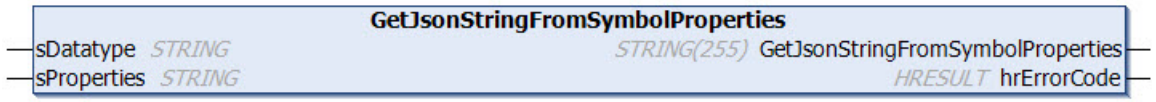
Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

```
sBuffer := fbJsonDataType.GetJsonStringFromSymbol('ST_Values',SIZEOF(stValues), ADR(stValues));
```

5.2.1.6.10 GetJsonStringFromSymbolProperties



Diese Methode erzeugt eine entsprechende JSON-Repräsentation von SPS-Attributen an einem Symbol. Im Unterschied zur Methode AddJsonKeyPropertiesFromSymbol [▶ 235] wird das Resultat nicht in eine Instanz vom Funktionsbaustein FB_JsonSaxWriter geschrieben, sondern in eine String-Variable. Als Eingangsparameter erhält die Methode den Datentypnamen des Symbols und eine String-Variable, welche eine durch Querbalken getrennte Liste der zu extrahierenden SPS-Attribute darstellt. Das Resultat wird direkt als Rückgabewert der Methode zurückgeliefert.

Die maximale Größe des von der Methode zurückgelieferten Strings ist 255 Zeichen. Bei längeren Zeichenfolgen wird von der Methode ein NULL-String zurückgegeben. In diesem Fall muss die Methode CopyJsonStringFromSymbolProperties [▶ 239]() verwendet werden.

Syntax

```
METHOD GetJsonStringFromSymbolProperties : STRING(255)
VAR_IN_OUT CONSTANT
  sDatatype    : STRING;
  sProperties : STRING;
END_VAR
```

```
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

Rückgabewert

Name	Typ
GetJsonStringFromSymbolProperties	STRING(255)

/ Ein-/Ausgänge

Name	Typ
sDatatype	STRING
sProperties	STRING

Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

```
sBuffer := fbJsonDataType.GetJsonStringFromSymbolProperties('ST_Values', 'Unit|DisplayName');
```

5.2.1.6.11 GetSizeJsonStringFromSymbol

Diese Methode liest die Größe der JSON-Repräsentanz eines Symbols aus. Der Wert wird dabei mit Nullterminierung angegeben.

Syntax

```
METHOD GetSizeJsonStringFromSymbol : UDINT
VAR_INPUT
  nData       :UDINT;
  pData       :PVOID;
END_VAR
VAR_IN_OUT CONSTANT
  sDatatype   : STRING;
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

Rückgabewert

Name	Typ
GetSizeJsonStringFromSymbol	UDINT

Eingänge

Name	Typ
nData	UDINT
pData	PVOID

/ Ein-/Ausgänge

Name	Typ
sDatatype	STRING

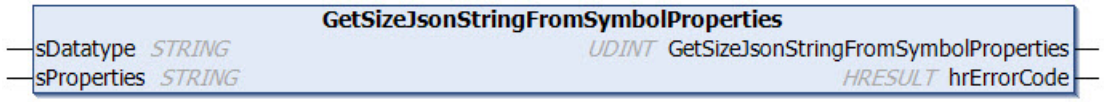
Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

```
nLen := fbJsonDataType.GetSizeJsonStringFromSymbol('BOOL',SIZEOF(bBool),ADR(bBool));
```

5.2.1.6.12 GetSizeJsonStringFromSymbolProperties



Diese Methode liest die Größe der JSON-Repräsentanz von SPS-Attributen an einem Symbol aus. Der Wert wird dabei mit Nullterminierung angegeben.

Syntax

```
METHOD GetSizeJsonStringFromSymbolProperties : UDINT
VAR_IN_OUT CONSTANT
  sDatatype   : STRING;
  sProperties : STRING;
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

Rückgabewert

Name	Typ
GetSizeJsonStringFromSymbolProperties	UDINT

/ Ein-/Ausgänge

Name	Typ
sDatatype	STRING
sProperties	STRING

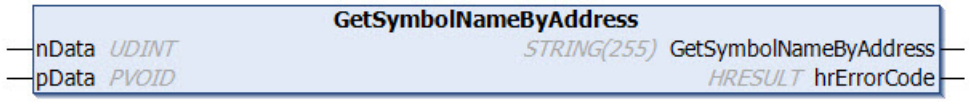
Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

```
nLen := fbJsonDataType.GetSizeJsonStringFromSymbolProperties('ST_Test','DisplayName|Unit');
```

5.2.1.6.13 GetSymbolNameByAddress



Diese Methode liefert den vollständigen (ADS-)Symbolnamen eines übergebenen Symbols.

Die maximale Größe des von der Methode zurückgelieferten Strings ist 255 Zeichen. Bei längeren Zeichenfolgen wird von der Methode ein Null-String zurückgegeben. In diesem Fall muss die Methode CopySymbolNameByAddress() [▶ 240] verwendet werden.

Syntax

```
METHOD GetSymbolNameByAddress : STRING(255)
VAR_INPUT
  nData       : UDINT;
  pData       : PVOID;
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

Rückgabewert

Name	Typ
GetSymbolNameByAddress	STRING(255)

Eingänge

Name	Typ
nData	UDINT
pData	PVOID

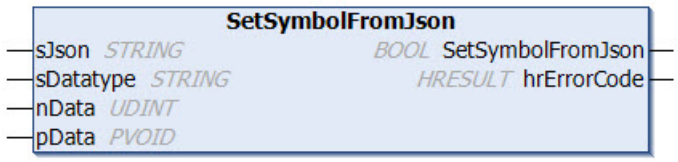
Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

```
sBuffer := fbJsonDataType.GetSymbolNameByAddress(SIZEOF(stValues), ADR(stValues));
```

5.2.1.6.14 SetSymbolFromJson



Diese Methode extrahiert einen String, der eine gültige JSON-Nachricht beinhaltet, und versucht die Inhalte des JSON-Objekts in eine äquivalente Datenstruktur zu speichern. Als Eingangsparameter erhält die Methode den String mit dem JSON-Objekt, den Datentypnamen der Zielstruktur sowie die Adresse und Größe der Zielstrukturinstanz.

Syntax

```
METHOD SetSymbolFromJson : BOOL
VAR_IN_OUT CONSTANT
  sJson       : STRING;
  sDatatype   : STRING;
END_VAR
VAR_INPUT
  nData       : UDINT;
  pData       : PVOID;
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

Rückgabewert

Name	Typ
SetSymbolFromJson	BOOL

Eingänge

Name	Typ
nData	UDINT
pData	PVOID

/ Ein-/Ausgänge

Name	Typ
sJson	STRING
sDatatype	STRING

Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

```
fbJsonDataType.SetSymbolFromJson(sJson,'ST_Values',SIZEOF(stValuesReceive), ADR(stValuesReceive));
```

5.2.1.7 FB_XmlDomParser

Strings im UTF-8-Format

Die hier verwendeten Variablen vom Typ STRING nutzen das UTF-8-Format. Diese STRING-Formatierung ist üblich bei IoT/MQTT-Kommunikation sowie JSON-Dokumenten.

Um Sonderzeichen und Texte verschiedenster Sprachen empfangen zu können, wird der Zeichensatz in den Bibliotheken Tc3_IotBase und Tc3_JsonXml nicht auf den typischen Zeichensatz vom Datentyp STRING beschränkt. Stattdessen wird der Unicode-Zeichensatz als UTF-8-Format in Verbindung mit dem Datentyp STRING verwendet.

Bei Verwendung des ASCII-Zeichensatzes besteht kein Unterschied zwischen der typischen Formatierung in einem STRING und der UTF-8-Formatierung eines STRING.

Weitere Informationen zum UTF-8-STRING-Format sowie vorhandenen Anzeige- und Konvertierungsmöglichkeiten finden Sie in der Dokumentation der SPS-Bibliothek Tc2_Utilities.

Ausgänge

Name	Typ
initStatus	HRESULT

Methoden

Name	Beschreibung
AppendAttribute [▶ 254]	Fügt ein neues Attribut zu einem existierenden Knoten hinzu.
AppendAttributeAsBool [▶ 254]	Fügt ein neues Attribut zu einem existierenden Knoten hinzu (Boolean).
AppendAttributeAsDouble [▶ 255]	Fügt ein neues Attribut zu einem existierenden Knoten hinzu (Double).
AppendAttributeAsFloat [▶ 256]	Fügt ein neues Attribut zu einem existierenden Knoten hinzu (Float).
AppendAttributeAsInt [▶ 256]	Fügt ein neues Attribut zu einem existierenden Knoten hinzu (Integer).
AppendAttributeAsLint [▶ 257]	Fügt ein neues Attribut zu einem existierenden Knoten hinzu (Integer64).
AppendAttributeAsUint [▶ 258]	Fügt ein neues Attribut zu einem existierenden Knoten hinzu (Unsigned Integer).
AppendAttributeAsUlint [▶ 259]	Fügt ein neues Attribut zu einem existierenden Knoten hinzu (Unsigned Integer64).
AppendAttributeCopy [▶ 259]	Fügt ein neues Attribut zu einem existierenden Knoten hinzu.
AppendChild [▶ 260]	Fügt einen neuen Knoten unterhalb eines existierenden Knotens ein.
AppendChildAsBool [▶ 261]	Fügt einen neuen Knoten unterhalb eines existierenden Knotens ein (Boolean).
AppendChildAsDouble [▶ 262]	Fügt einen neuen Knoten unterhalb eines existierenden Knotens ein (Double).
AppendChildAsFloat [▶ 262]	Fügt einen neuen Knoten unterhalb eines existierenden Knotens ein (Float).
AppendChildAsInt [▶ 263]	Fügt einen neuen Knoten unterhalb eines existierenden Knotens ein (Integer).
AppendChildAsLint [▶ 264]	Fügt einen neuen Knoten unterhalb eines existierenden Knotens ein (Integer64).
AppendChildAsUint [▶ 265]	Fügt einen neuen Knoten unterhalb eines existierenden Knotens ein (Unsigned Integer).
AppendChildAsUlint [▶ 265]	Fügt einen neuen Knoten unterhalb eines existierenden Knotens ein (Unsigned Integer64).
AppendCopy [▶ 266]	Fügt einen neuen Knoten unterhalb eines existierenden Knotens ein.
AppendNode [▶ 267]	Fügt einen neuen Knoten unterhalb eines existierenden Knotens ein.
Attribute [▶ 267]	Liest das Attribut eines gegebenen XML-Knotens aus.
AttributeAsBool [▶ 268]	Liefert den Wert eines Attributs als Datentyp Boolean.
AttributeAsDouble [▶ 268]	Liefert den Wert eines Attributs als Datentyp Double.
AttributeAsFloat [▶ 269]	Liefert den Wert eines Attributs als Datentyp Float.
AttributeAsInt [▶ 269]	Liefert den Wert eines Attributs als Datentyp Integer.
AttributeAsLint [▶ 270]	Liefert den Wert eines Attributs als Datentyp Integer64.
AttributeAsUint [▶ 270]	Liefert den Wert eines Attributs als Datentyp Unsigned Integer.
AttributeAsUlint [▶ 271]	Liefert den Wert eines Attributs als Datentyp Unsigned Integer64.
AttributeBegin [▶ 271]	Liefert einen Iterator über alle Attribute eines XML-Knotens.
AttributeFromIterator [▶ 272]	Konvertiert die aktuelle Position eines Iterators in ein XML-Attribut-Objekt.
AttributeName [▶ 273]	Liefert den Namen eines gegebenen Attributs.
Attributes [▶ 273]	Dient zur Navigation durch den DOM und liefert einen Iterator auf alle gefundenen Attribute an einem XML-Knoten zurück.
AttributeText [▶ 274]	Liefert den Text eines gegebenen Attributs.
Begin [▶ 274]	Liefert einen Iterator über alle Kindelemente eines XML-Knotens.
BeginByName [▶ 275]	Liefert einen Iterator über alle Kindelemente eines XML-Knotens, beginnend ab einem bestimmten Element.
Child [▶ 275]	Dient zur Navigation durch den DOM und liefert eine Referenz auf das (erste) Kindelement des aktuellen Knotens zurück.

Name	Beschreibung
ChildByAttribute [▶ 276]	Dient zur Navigation durch den DOM und liefert eine Referenz auf ein Kindelement im XML-Dokument zurück.
ChildByAttributeAndName [▶ 277]	Dient zur Navigation durch den DOM und liefert eine Referenz auf ein Kindelement im XML-Dokument zurück.
ChildByName [▶ 277]	Dient zur Navigation durch den DOM und liefert eine Referenz auf ein Kindelement im XML-Dokument zurück.
Children [▶ 278]	Dient zur Navigation durch den DOM und liefert einen Iterator auf mehrere gefundene Kindelemente im XML-Dokument zurück.
ChildrenByName [▶ 279]	Dient zur Navigation durch den DOM und liefert einen Iterator auf mehrere gefundene Kindelemente im XML-Dokument zurück.
ClearIterator [▶ 279]	Diese Methode setzt bereits verwendete Iteratoren zurück, sodass diese bei dem nächsten Programmdurchlauf wieder verwendet werden können.
Compare [▶ 280]	Überprüft zwei Iteratoren auf Gleichheit.
CopyAttributeText [▶ 281]	Liest den Wert eines XML-Attributs aus und schreibt diesen in eine Variable vom Datentyp String.
CopyDocument [▶ 281]	Kopiert den Inhalt des DOM-Speichers in eine Variable vom Datentyp String.
CopyNodeText [▶ 282]	Liest den Wert eines XML-Knotens aus und schreibt diesen in eine Variable vom Datentyp String.
CopyNodeXml [▶ 283]	Liest die XML-Struktur eines XML-Knotens aus und schreibt diese in eine Variable vom Datentyp String.
End	
FirstNodeByPath [▶ 283]	Navigiert anhand eines an die Methode übergebenen Pfads durch ein XML-Dokument.
GetAttributeTextLength [▶ 284]	Liefert die Länge des Werts eines XML-Attributs.
GetDocumentLength [▶ 285]	Liefert die Länge eines XML-Dokuments in Bytes.
GetDocumentNode [▶ 285]	Liefert den Root-Knoten eines XML-Dokuments.
GetNodeTextLength [▶ 285]	Liefert die Länge des Werts eines XML-Knotens.
GetNodeXmlLength [▶ 286]	Liefert die Länge der XML-Struktur eines XML-Knotens.
GetRootNode [▶ 286]	Liefert eine Referenz zum ersten XML-Knoten im XML-Dokument.
InsertAttributeCopy [▶ 287]	Fügt ein Attribut zu einem XML-Knoten hinzu und kopiert hierbei den Namen und den Wert eines existierenden Attributs.
InsertAttribute [▶ 287]	Fügt ein Attribut zu einem XML-Knoten hinzu.
InsertChild [▶ 288]	Fügt einen Knoten zu einem existierenden XML-Knoten hinzu.
InsertCopy [▶ 289]	Fügt einen neuen Knoten zu einem existierenden XML Knoten hinzu und kopiert hierbei einen existierenden Knoten.
IsEnd [▶ 289]	Überprüft, ob sich ein gegebener XML-Iterator am Ende der zu durchlaufenden Iteration befindet.
LoadDocumentFromFile [▶ 290]	Lädt ein XML-Dokument aus einer Datei.
NewDocument [▶ 291]	Erzeugt ein leeres XML-Dokument im DOM-Speicher.
Next [▶ 291]	Setzt einen XML-Iterator auf das nächste zu durchlaufende Objekt.
NextAttribute [▶ 292]	Liefert zu einem gegebenen XML-Attribut das nächste Attribut.
NextByName [▶ 292]	Setzt einen XML-Iterator auf das nächste zu durchlaufende Objekt, das anhand seines Namens identifiziert wird.
NextSibling [▶ 293]	Liefert zu einem gegebenen XML-Knoten den nächsten direkten Knoten auf derselben XML-Ebene.
NextSiblingByName [▶ 293]	Liefert zu einem gegebenen XML-Knoten den nächsten direkten Knoten mit einem bestimmten Namen auf derselben XML-Ebene.
Node [▶ 294]	Wird in Zusammenhang mit einem Iterator verwendet, um durch den DOM zu navigieren.

Name	Beschreibung
NodeAsBool [▶ 295]	Liefert den Text eines XML-Knotens als Datentyp Boolean.
NodeAsDouble [▶ 295]	Liefert den Text eines XML-Knotens als Datentyp Double.
NodeAsFloat [▶ 296]	Liefert den Text eines XML-Knotens als Datentyp Float.
NodeAsInt [▶ 296]	Liefert den Text eines XML-Knotens als Datentyp Integer.
NodeAsLint [▶ 297]	Liefert den Text eines XML-Knotens als Datentyp Integer64.
NodeAsUint [▶ 297]	Liefert den Text eines XML-Knotens als Datentyp Unsigned Integer.
NodeAsUlint [▶ 298]	Liefert den Text eines XML-Knotens als Datentyp Unsigned Integer64.
NodeName [▶ 298]	Gibt den Namen eines XML-Knotens zurück.
NodeText [▶ 299]	Liefert den Text eines XML-Knotens.
ParseDocument [▶ 299]	Lädt ein XML-Dokument zur weiteren Verarbeitung in den DOM-Speicher.
RemoveAttribute	
RemoveAttributeByName	
RemoveChild [▶ 300]	entfernt einen XML-Kindknoten von einem gegebenen XML-Knoten.
RemoveChildByName [▶ 300]	Entfernt einen XML-Kindknoten von einem gegebenen XML-Knoten.
SaveDocumentToFile [▶ 301]	Speichert das aktuelle XML-Dokument in einer Datei.
SetAttribute [▶ 302]	Setzt den Wert eines Attributs.
SetAttributeAsBool [▶ 303]	Setzt den Wert eines Attributs (Boolean).
SetAttributeAsDouble [▶ 303]	Setzt den Wert eines Attributs (Double).
SetAttributeAsFloat [▶ 304]	Setzt den Wert eines Attributs (Float).
SetAttributeAsInt [▶ 304]	Setzt den Wert eines Attributs (Integer).
SetAttributeAsLint [▶ 305]	Setzt den Wert eines Attributs (Integer64).
SetAttributeAsUint [▶ 305]	Setzt den Wert eines Attributs (Unsigned Integer).
SetAttributeAsUlint [▶ 306]	Setzt den Wert eines Attributs (Unsigned Integer64).
SetChild [▶ 306]	Setzt den Wert eines XML-Knotens (String).
SetChildAsBool [▶ 307]	Setzt den Wert eines XML-Knotens (Boolean).
SetChildAsDouble [▶ 308]	Setzt den Wert eines XML-Knotens (Double).
SetChildAsFloat [▶ 308]	Setzt den Wert eines XML-Knotens (Float).
SetChildAsInt [▶ 309]	Setzt den Wert eines XML-Knotens (Integer).
SetChildAsLint [▶ 309]	Setzt den Wert eines XML-Knotens (Integer64).
SetChildAsUint [▶ 310]	Setzt den Wert eines XML-Knotens (Unsigned Integer).
SetChildAsUlint [▶ 310]	Setzt den Wert eines XML-Knotens (Unsigned Integer64).

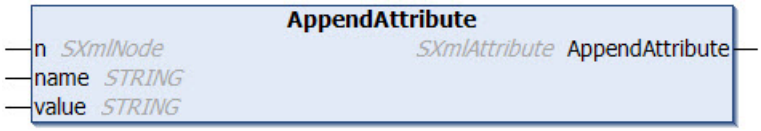
Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4022	x86, x64, ARM	Tc3_JsonXml

Sehen Sie dazu auch

- AppendAttribute [▶ 254]
- AppendAttributeAsBool [▶ 254]
- AppendAttributeAsDouble [▶ 255]
- AppendAttributeAsFloat [▶ 256]
- AppendAttributeAsInt [▶ 256]
- AppendAttributeAsLint [▶ 257]
- AppendAttributeAsUint [▶ 258]

5.2.1.7.1 AppendAttribute



Diese Methode fügt ein neues Attribut zu einem existierenden Knoten hinzu. Der Name und der Wert des neuen Attributs sowie der existierende XML-Knoten werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf das neu hinzugefügte Attribut.

Syntax

```
METHOD AppendAttribute : SXmlAttribute
VAR_INPUT
  n     : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name  : STRING;
  value : STRING;
END_VAR
```

Rückgabewert

Name	Typ
AppendAttribute	SXmlAttribute

Eingänge

Name	Typ
n	SXmlNode

Ein-/Ausgänge

Name	Typ
name	STRING
value	STRING

Beispielaufruf:

```
objAttribute := fbXml.AppendAttribute(objMachine, 'Name', 'some value');
```

5.2.1.7.2 AppendAttributeAsBool

Diese Methode fügt ein neues Attribut zu einem existierenden Knoten hinzu. Der Wert des Attributs ist hierbei vom Datentyp Boolean. Der Name und der Wert des neuen Attributs sowie der existierende XML-Knoten werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf das neu hinzugefügte Attribut.

Syntax

```
METHOD AppendAttributeAsBool : SXmlAttribute
VAR_INPUT
  n     : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name  : STRING;
END_VAR
```

```
VAR_INPUT
  value : BOOL;
END_VAR
```

Rückgabewert

Name	Typ
AppendAttributeAsBool	SXmlAttribute

Eingänge

Name	Typ
n	SXmlNode
value	BOOL

/ Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

```
objAttribute := fbXml.AppendAttributeAsBool(objMachine, 'Name', TRUE);
```

5.2.1.7.3 AppendAttributeAsDouble

Diese Methode fügt ein neues Attribut zu einem existierenden Knoten hinzu. Der Wert des Attributs ist hierbei vom Datentyp Double. Der Name und der Wert des neuen Attributs sowie der existierende XML-Knoten werden der Methode als Eingangsparameter übergeben. Die Methode returniert eine Referenz auf das neu hinzugefügte Attribut.

Syntax

```
METHOD AppendAttributeAsDouble : SXmlAttribute
VAR_INPUT
  n     : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name  : STRING;
END_VAR
VAR_INPUT
  value : LREAL;
END_VAR
```

Rückgabewert

Name	Typ
AppendAttributeAsDouble	SXmlAttribute

Eingänge

Name	Typ
n	SXmlNode
value	LREAL

/ Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

```
objAttribute := fbXml.AppendAttributeAsDouble(objMachine, 'Name', 42.42);
```

5.2.1.7.4 AppendAttributeAsFloat

Diese Methode fügt ein neues Attribut zu einem existierenden Knoten hinzu. Der Wert des Attributs ist hierbei vom Datentyp Float. Der Name und der Wert des neuen Attributs sowie der existierende XML-Knoten werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf das neu hinzugefügte Attribut.

Syntax

```
METHOD AppendAttributeAsFloat : SXmlAttribute
VAR_INPUT
  n     : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name  : STRING;
END_VAR
VAR_INPUT
  value : REAL;
END_VAR
```

Rückgabewert

Name	Typ
AppendAttributeAsFloat	SXmlAttribute

Eingänge

Name	Typ
n	SXmlNode
value	REAL

/ Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

```
objAttribute := fbXml.AppendAttributeAsFloat(objMachine, 'Name', 42.42);
```

5.2.1.7.5 AppendAttributeAsInt

Diese Methode fügt ein neues Attribut zu einem existierenden Knoten hinzu. Der Wert des Attributs ist hierbei vom Datentyp Integer. Der Name und der Wert des neuen Attributs sowie der existierende XML-Knoten werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf das neu hinzugefügte Attribut.

Syntax

```
METHOD AppendAttributeAsInt : SXmlAttribute
VAR_INPUT
  n     : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name  : STRING;
END_VAR
VAR_INPUT
  value : DINT;
END_VAR
```

Rückgabewert

Name	Typ
AppendAttributeAsInt	SXmlAttribute

Eingänge

Name	Typ
n	SXmlNode
value	DINT

/ Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

```
objAttribute := fbXml.AppendAttributeAsInt(objMachine, 'Name', 42);
```

5.2.1.7.6 AppendAttributeAsLint

Diese Methode fügt ein neues Attribut zu einem existierenden Knoten hinzu. Der Wert des Attributs ist hierbei vom Datentyp Integer64. Der Name und der Wert des neuen Attributs sowie der existierende XML-Knoten werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf das neu hinzugefügte Attribut.

Syntax

```
METHOD AppendAttributeAsLint : SXmlAttribute
VAR_INPUT
  n     : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name  : STRING;
END_VAR
VAR_INPUT
  value : LINT;
END_VAR
```

Rückgabewert

Name	Typ
AppendAttributeAsLint	SXmlAttribute

Eingänge

Name	Typ
n	SXmlNode
value	LINT

/ Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

```
objAttribute := fbXml.AppendAttributeAsLint(objMachine, 'Name', 42);
```

5.2.1.7.7 AppendAttributeAsUint

Diese Methode fügt ein neues Attribut zu einem existierenden Knoten hinzu. Der Wert des Attributs ist hierbei vom Datentyp Unsigned Integer. Der Name und der Wert des neuen Attributs sowie der existierende XML-Knoten werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf das neu hinzugefügte Attribut.

Syntax

```
METHOD AppendAttributeAsUint : SXmlAttribute
VAR_INPUT
  n     : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name  : STRING;
END_VAR
VAR_INPUT
  value : UDINT;
END_VAR
```

Rückgabewert

Name	Typ
AppendAttributeAsUint	SXmlAttribute

Eingänge

Name	Typ
n	SXmlNode
value	UDINT

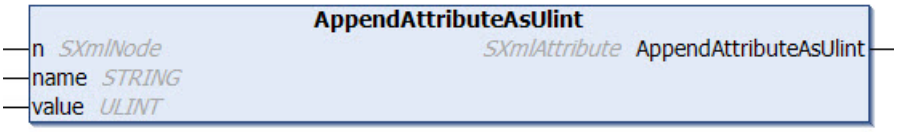
/ Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

```
objAttribute := fbXml.AppendAttributeAsUint(objMachine, 'Name', 42);
```

5.2.1.7.8 AppendAttributeAsUlint



Diese Methode fügt ein neues Attribut zu einem existierenden Knoten hinzu. Der Wert des Attributs ist hierbei vom Datentyp Unsigned Integer64. Der Name und der Wert des neuen Attributs sowie der existierende XML-Knoten werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf das neu hinzugefügte Attribut.

Syntax

```
METHOD AppendAttributeAsUlint : SXmlAttribute
VAR_INPUT
  n     : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name  : STRING;
END_VAR
VAR_INPUT
  value : ULINT;
END_VAR
```

Rückgabewert

Name	Typ
AppendAttributeAsUlint	SXmlAttribute

Eingänge

Name	Typ
n	SXmlNode
value	ULINT

/ Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

```
objAttribute := fbXml.AppendAttributeAsUlint(objMachine, 'Name', 42);
```

5.2.1.7.9 AppendAttributeCopy

Diese Methode fügt ein neues Attribut zu einem existierenden Knoten hinzu. Der Name und der Wert des neuen Attributs werden hierbei von einem existierenden Attribut übernommen bzw. kopiert. Das existierende Attribut wird der Methode als Eingangsparameter übergeben.

Syntax

```
METHOD AppendAttributeCopy : SXmlAttribute
INPUT_VAR
  n    : SXmlNode;
  copy : SXmlAttribute;
END_VAR
```

Rückgabewert

Name	Typ
AppendAttributeCopy	SXmlAttribute

Eingänge

Name	Typ
n	SXmlNode
copy	SXmlAttribute

Beispielaufruf:

```
xmlNewAttribute := fbXml.AppendAttributeCopy(xmlNode, xmlExistingAttribute);
```

5.2.1.7.10 AppendChild

Diese Methode fügt einen neuen Knoten unterhalb eines existierenden Knotens ein. Der Wert des neuen Knotens ist hierbei vom Datentyp STRING. Der Name und der Wert des neuen Knotens sowie eine Referenz auf den existierenden Knoten werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf den neu hinzugefügten Knoten. Der Eingangsparameter cdata gibt an, ob der Wert des Knotens in einem CDATA-Block gekapselt werden soll, sodass bestimmte Sonderzeichen wie z. B. "<" und ">" als Werte erlaubt sind.

Syntax

```
METHOD AppendChild : SXmlNode
VAR_INPUT
  n     : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name  : STRING;
  value : STRING;
END_VAR
VAR_INPUT
  cdata : BOOL;
END_VAR
```

Rückgabewert

Name	Typ
AppendChild	SXmlNode

Eingänge

Name	Typ
n	SXmlNode
cdata	BOOL

/ Ein-/Ausgänge

Name	Typ
name	STRING
value	STRING

Beispielaufruf:

```
xmlNewNode := fbXml.AppendChild(xmlExisting, 'Controller', 'CX5120', FALSE);
```

5.2.1.7.11 AppendChildAsBool

Diese Methode fügt einen neuen Knoten unterhalb eines existierenden Knotens ein. Der Wert des neuen Knotens ist hierbei vom Datentyp Boolean. Der Name und der Wert des neuen Knotens sowie eine Referenz auf den existierenden Knoten werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf den neu hinzugefügten Knoten.

Syntax

```
METHOD AppendChildAsBool : SXmlNode
VAR_INPUT
  n     : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name  : STRING;
END_VAR
VAR_INPUT
  value : BOOL;
END_VAR
```

Rückgabewert

Name	Typ
AppendChildAsBool	SXmlNode

Eingänge

Name	Typ
n	SXmlNode
value	BOOL

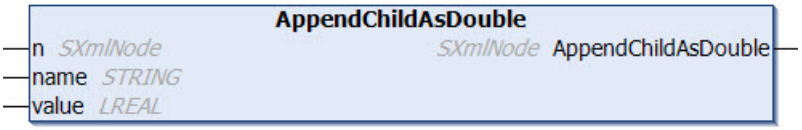
/ Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

```
xmlNewNode := fbXml.AppendChildAsBool(xmlExisting, 'SomeName', TRUE);
```

5.2.1.7.12 AppendChildAsDouble



Diese Methode fügt einen neuen Knoten unterhalb eines existierenden Knotens ein. Der Wert des neuen Knotens ist hierbei vom Datentyp Double. Der Name und der Wert des neuen Knotens sowie eine Referenz auf den existierenden Knoten werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf den neu hinzugefügten Knoten.

Syntax

```
METHOD AppendChildAsDouble : SXmlNode
VAR_INPUT
  n     : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name  : STRING;
END_VAR
VAR_INPUT
  value : LREAL;
END_VAR
```

Rückgabewert

Name	Typ
AppendChildAsDouble	SXmlNode

Eingänge

Name	Typ
n	SXmlNode
value	LREAL

Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

```
xmlNewNode := fbXml.AppendChildAsDouble(xmlExisting, 'SomeName', 42.42);
```

5.2.1.7.13 AppendChildAsFloat

Diese Methode fügt einen neuen Knoten unterhalb eines existierenden Knotens ein. Der Wert des neuen Knotens ist hierbei vom Datentyp Float. Der Name und der Wert des neuen Knotens sowie eine Referenz auf den existierenden Knoten werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf den neu hinzugefügten Knoten.

Syntax

```
METHOD AppendChildAsFloat : SXmlNode
VAR_INPUT
  n     : SXmlNode;
END_VAR
```

```
VAR_IN_OUT CONSTANT
  name  : STRING;
END_VAR
VAR_INPUT
  value : REAL;
END_VAR
```

Rückgabewert

Name	Typ
AppendChildAsFloat	SXmlNode

Eingänge

Name	Typ
n	SXmlNode
value	REAL

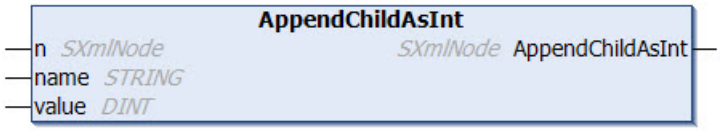
/ Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

```
xmlNewNode := fbXml.AppendChildAsFloat(xmlExisting, 'SomeName', 42.42);
```

5.2.1.7.14 AppendChildAsInt



Diese Methode fügt einen neuen Knoten unterhalb eines existierenden Knotens ein. Der Wert des neuen Knotens ist hierbei vom Datentyp Integer. Der Name und der Wert des neuen Knotens sowie eine Referenz auf den existierenden Knoten werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf den neu hinzugefügten Knoten.

Syntax

```
METHOD AppendChildAsInt : SXmlNode
VAR_INPUT
  n     : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name  : STRING;
END_VAR
VAR_INPUT
  value : DINT;
END_VAR
```

Rückgabewert

Name	Typ
AppendChildAsInt	SXmlNode

Eingänge

Name	Typ
n	SXmlNode
value	DINT

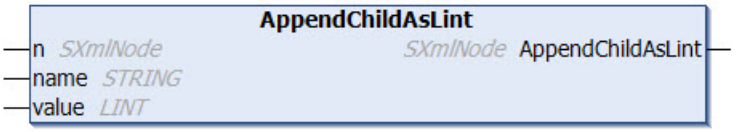
/ Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

```
xmlNewNode := fbXml.AppendChildAsInt(xmlExisting, 'SomeName', 42);
```

5.2.1.7.15 AppendChildAsLint



Diese Methode fügt einen neuen Knoten unterhalb eines existierenden Knotens ein. Der Wert des neuen Knotens ist hierbei vom Datentyp Integer64. Der Name und der Wert des neuen Knotens sowie eine Referenz auf den existierenden Knoten werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf den neu hinzugefügten Knoten.

Syntax

```
METHOD AppendChildAsLint : SXmlNode
VAR_INPUT
  n     : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name  : STRING;
END_VAR
VAR_INPUT
  value : LINT;
END_VAR
```

Rückgabewert

Name	Typ
AppendChildAsLint	SXmlNode

Eingänge

Name	Typ
n	SXmlNode
value	LINT

/ Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

```
xmlNewNode := fbXml.AppendChildAsLint(xmlExisting, 'SomeName', 42);
```

5.2.1.7.16 AppendChildAsUint

Diese Methode fügt einen neuen Knoten unterhalb eines existierenden Knotens ein. Der Wert des neuen Knotens ist hierbei vom Datentyp Unsigned Integer. Der Name und der Wert des neuen Knotens sowie eine Referenz auf den existierenden Knoten werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf den neu hinzugefügten Knoten.

Syntax

```
METHOD AppendChildAsUint : SXmlNode
VAR_INPUT
  n     : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name  : STRING;
END_VAR
VAR_INPUT
  value : UDINT;
END_VAR
```

Rückgabewert

Name	Typ
AppendChildAsUint	SXmlNode

Eingänge

Name	Typ
n	SXmlNode
value	UDINT

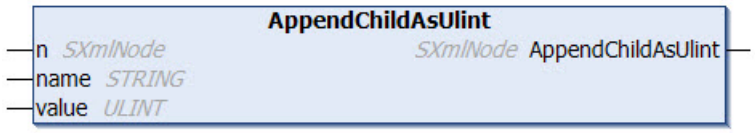
Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

```
xmlNewNode := fbXml.AppendChildAsUint(xmlExisting, 'SomeName', 42);
```

5.2.1.7.17 AppendChildAsUlint



Diese Methode fügt einen neuen Knoten unterhalb eines existierenden Knotens ein. Der Wert des neuen Knotens ist hierbei vom Datentyp Unsigned Integer64. Der Name und der Wert des neuen Knotens sowie eine Referenz auf den existierenden Knoten werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf den neu hinzugefügten Knoten.

Syntax

```
METHOD AppendChildAsUlint : SXmlNode
VAR_INPUT
  n     : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
```

```
  name  : STRING;
END_VAR
VAR_INPUT
  value : ULINT;
END_VAR
```

Rückgabewert

Name	Typ
AppendChildAsUlint	SXmlNode

Eingänge

Name	Typ
n	SXmlNode
value	ULINT

/ Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

```
xmlNewNode := fbXml.AppendChildAsUlint(xmlExisting, 'SomeName', 42);
```

5.2.1.7.18 AppendCopy

Diese Methode fügt einen neuen Knoten unterhalb eines existierenden Knotens ein. Der Name und der Wert des neuen Knotens werden von einem existierenden Knoten übernommen bzw. kopiert. Die Referenzen auf die existierenden Knoten werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf den neu hinzugefügten Knoten.

Syntax

```
METHOD AppendCopy : SXmlNode
VAR_INPUT
  n    : SXmlNode;
  copy : SXmlNode;
END_VAR
```

Rückgabewert

Name	Typ
AppendCopy	SXmlNode

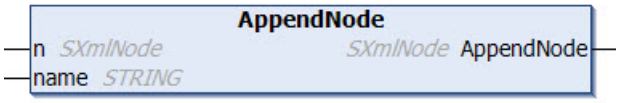
Eingänge

Name	Typ
n	SXmlNode
copy	SXmlNode

Beispielaufruf:

```
xmlNewNode := fbXml.AppendCopy(xmlParentNode, xmlExistingNode);
```

5.2.1.7.19 AppendNode



Diese Methode fügt einen neuen Knoten zu einem existierenden Knoten hinzu. Der existierende Knoten sowie der Name des neuen Knotens werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf den neu angefügten Knoten.

Syntax

```
METHOD AppendNode : SXmlNode
VAR_INPUT
  n    : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name : STRING;
END_VAR
```

Rückgabewert

Name	Typ
AppendNode	SXmlNode

Eingänge

Name	Typ
n	SXmlNode

/ Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

```
objMachines := fbXml.AppendNode(objRoot, 'Machines');
```

5.2.1.7.20 Attribute

Mit dieser Methode kann das Attribut eines gegebenen XML-Knotens ausgelesen werden. Der XML-Knoten und der Name des Attributs werden der Methode als Eingangsparameter übergeben. Nach Aufruf der Methode müssen weitere Methoden aufgerufen werden, um z. B. den Wert des Attributs auszulesen, z. B. AttributeAsInt().

Syntax

```
METHOD Attribute : SXmlAttribute
VAR_INPUT
  n    : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name : STRING;
END_VAR
```

Rückgabewert

Name	Typ
Attribute	SXmlAttribute

Eingänge

Name	Typ
n	SXmlNode

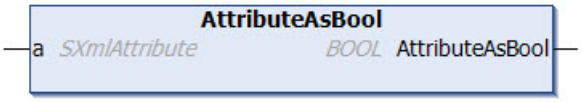
/ Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

```
xmlMachine1Attribute := fbXml.Attribute(xmlMachine1, 'Type');
```

5.2.1.7.21 AttributeAsBool



Diese Methode liefert den Wert eines Attributs als Datentyp Boolean. Das Attribut wird der Methode als Eingangsparameter übergeben.

Syntax

```
METHOD AttributeAsBool : BOOL
VAR_INPUT
  a : SXmlAttribute;
END_VAR
```

Rückgabewert

Name	Typ
AttributeAsBool	BOOL

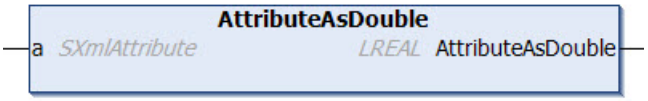
Eingänge

Name	Typ
a	SXmlAttribute

Beispielaufruf:

```
bValue := fbXml.AttributeAsBool(xmlAttr);
```

5.2.1.7.22 AttributeAsDouble



Diese Methode liefert den Wert eines Attributs als Datentyp Double. Das Attribut wird der Methode als Eingangsparameter übergeben.

Syntax

```
METHOD AttributeAsDouble : LREAL
VAR_INPUT
  a : SXmlAttribute;
END_VAR
```

Rückgabewert

Name	Typ
AttributeAsDouble	LREAL

Eingänge

Name	Typ
a	SXmlAttribute

Beispielaufruf:

```
lrValue := fbXml.AttributeAsDouble(xmlAttr);
```

5.2.1.7.23 AttributeAsFloat

Diese Methode liefert den Wert eines Attributs als Datentyp Float. Das Attribut wird der Methode als Eingangsparameter übergeben.

Syntax

```
METHOD AttributeAsFloat: REAL
VAR_INPUT
  a : SXmlAttribute;
END_VAR
```

Rückgabewert

Name	Typ
AttributeAsFloat	REAL

Eingänge

Name	Typ
a	SXmlAttribute

Beispielaufruf:

```
rValue := fbXml.AttributeAsFloat(xmlAttr);
```

5.2.1.7.24 AttributeAsInt

Diese Methode liefert den Wert eines Attributs als Datentyp Integer. Das Attribut wird der Methode als Eingangsparameter übergeben.

Syntax

```
METHOD AttributeAsInt : DINT
VAR_INPUT
  a : SXmlAttribute;
END_VAR
```

Rückgabewert

Name	Typ
AttributeAsInt	DINT

Eingänge

Name	Typ
a	SXmlAttribute

Beispielaufruf:

```
nValue := fbXml.AttributeAsInt(xmlAttr);
```

5.2.1.7.25 AttributeAsLint

Diese Methode liefert den Wert eines Attributs als Datentyp Integer64. Das Attribut wird der Methode als Eingangsparameter übergeben.

Syntax

```
METHOD AttributeAsLint : LINT
VAR_INPUT
  a : SXmlAttribute;
END_VAR
```

Rückgabewert

Name	Typ
AttributeAsLint	LINT

Eingänge

Name	Typ
a	SXmlAttribute

Beispielaufruf:

```
nValue := fbXml.AttributeAsLint(xmlAttr);
```

5.2.1.7.26 AttributeAsUint

Diese Methode liefert den Wert eines Attributs als Datentyp Unsigned Integer. Das Attribut wird der Methode als Eingangsparameter übergeben.

Syntax

```
METHOD AttributeAsUint : UDINT
VAR_INPUT
  a : SXmlAttribute;
END_VAR
```

Rückgabewert

Name	Typ
AttributeAsUint	UDINT

Eingänge

Name	Typ
a	SXmlAttribute

Beispielaufruf:

```
nValue := fbXml.AttributeAsUint(xmlAttr);
```

5.2.1.7.27 AttributeAsUlint

Diese Methode liefert den Wert eines Attributs als Datentyp Unsigned Integer64. Das Attribut wird der Methode als Eingangsparameter übergeben.

Syntax

```
METHOD AttributeAsUlint : ULINT
VAR_INPUT
  a : SXmlAttribute;
END_VAR
```

Rückgabewert

Name	Typ
AttributeAsUlint	ULINT

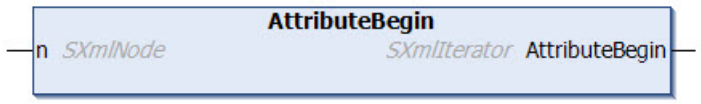
Eingänge

Name	Typ
a	SXmlAttribute

Beispielaufruf:

```
nValue := fbXml.AttributeAsUlint(xmlAttr);
```

5.2.1.7.28 AttributeBegin



Diese Methode liefert einen Iterator über alle Attribute eines XML-Knotens. Der XML-Knoten wird der Methode als Eingangsparameter übergeben.

Syntax

```
METHOD AttributeBegin : SXmlIterator
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Rückgabewert

Name	Typ
AttributeBegin	SXmlIterator

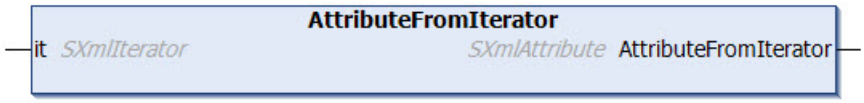
Eingänge

Name	Typ
n	SXmlNode

Beispielaufruf:

```
xmlIterator := fbXml.AttributeBegin(xmlNode);
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlAttr     := fbXml.AttributeFromIterator(xmlIterator);
  nAttrValue  := fbXml.AttributeAsInt(xmlAttr);
  xmlIterator := fbXml.Next(xmlIterator);
END_WHILE
```

5.2.1.7.29 AttributeFromIterator



Diese Methode konvertiert die aktuelle Position eines Iterators in ein XML-Attribut-Objekt. Der Iterator wird der Methode als Eingangsparameter übergeben.

Syntax

```
METHOD AttributeFromIterator : SXmlAttribute
VAR_INPUT
  it : SXmlIterator;
END_VAR
```

Rückgabewert

Name	Typ
AttributeFromIterator	SXmlAttribute

Eingänge

Name	Typ
it	SXmlIterator

Beispielaufruf:

```
xmlIterator := fbXml.AttributeBegin(xmlNode);
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlAttr     := fbXml.AttributeFromIterator(xmlIterator);
  nAttrValue  := fbXml.AttributeAsInt(xmlAttr);
  xmlIterator := fbXml.Next(xmlIterator);
END_WHILE
```

5.2.1.7.30 AttributeName

Diese Methode liefert den Namen eines gegebenen Attributs. Das Attribut wird der Methode als Eingangsparameter übergeben.

Syntax

```
METHOD AttributeName : STRING
VAR_INPUT
  a : SXmlAttribute;
END_VAR
```

Rückgabewert

Name	Typ
AttributeName	STRING

Eingänge

Name	Typ
a	SXmlAttribute

Beispielaufruf:

```
sName := fbXml.AttributeName(xmlAttr);
```

5.2.1.7.31 Attributes

Diese Methode dient zur Navigation durch den DOM und liefert einen Iterator auf alle gefundenen Attribute an einem XML-Knoten zurück. Der Iterator kann anschließend zur weiteren Navigation durch die gefundenen Elemente verwendet werden. Der Knoten und eine Referenz auf den Iterator werden der Methode als Eingangsparameter übergeben.

Syntax

```
METHOD Attributes : SXmlAttribute
VAR_INPUT
  n : SXmlNode;
  it : REFERENCE TO SXmlIterator;
END_VAR
```

Rückgabewert

Name	Typ
Attributes	SXmlAttribute

Eingänge

Name	Typ
it	REFERENCE TO SXmlIterator

Beispielaufruf:

```
xmlRet := fbXml.Attributes(xmlNode, xmlIterator);
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlMachineAttrRef  := fbXml.Attribute(xmlIterator);
  xmlMachineAttrText := fbXml.AttributeText(xmlMachineAttrRef);
  xmlIterator        := fbXml.Next(xmlIterator);
END_WHILE
```

5.2.1.7.32 AttributeText

Diese Methode liefert den Text eines gegebenen Attributs. Das Attribut wird der Methode als Eingangsparameter übergeben.

Syntax

```
METHOD AttributeText : STRING(255)
VAR_INPUT
  a : SXmlAttribute;
END_VAR
```

Rückgabewert

Name	Typ
AttributeText	STRING(255)

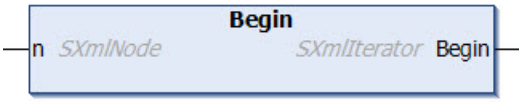
Eingänge

Name	Typ
a	SXmlAttribute

Beispielaufruf:

```
sText := fbXml.AttributeText(xmlAttr);
```

5.2.1.7.33 Begin



Diese Methode liefert einen Iterator über alle Kindelemente eines XML-Knotens, immer beginnend ab dem ersten Kindelement. Der XML-Knoten wird der Methode als Eingangsparameter übergeben.

Syntax

```
METHOD Begin : SXmlIterator
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Rückgabewert

Name	Typ
Begin	SXmlIterator

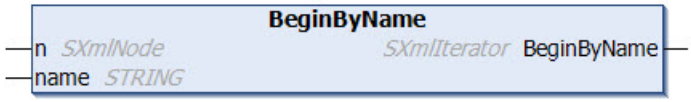
Eingänge

Name	Typ
n	SXmlNode

Beispielaufruf:

```
xmlIterator := fbXml.Begin(xmlNode);
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlNodeRef   := fbXml.Node(xmlIterator);
  xmlNodeValue := fbXml.NodeText(xmlNodeRef);
  xmlIterator  := fbXml.Next(xmlIterator);
END_WHILE
```

5.2.1.7.34 BeginByName



Diese Methode liefert einen Iterator über alle Kindelemente eines XML-Knotens, beginnend ab einem bestimmten Element. Der XML-Knoten wird der Methode als Eingangsparameter übergeben.

Syntax

```
METHOD BeginByName : SXmlIterator
VAR_INPUT
  n    : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name : STRING;
END_VAR
```

Rückgabewert

Name	Typ
BeginByName	SXmlIterator

Eingänge

Name	Typ
n	SXmlNode

/ Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

```
xmlNode := fbXml.ChildByName(xmlDoc, 'Machines');
xmlIterator    := fbXml.BeginByName(xmlNode, 'NameX');
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlNodeRef   := fbXml.Node(xmlIterator);
  xmlNodeValue := fbXml.NodeText(xmlNodeRef);
  xmlIterator  := fbXml.Next(xmlIterator);
END_WHILE
```

5.2.1.7.35 Child

Diese Methode dient zur Navigation durch den DOM und liefert eine Referenz auf das (erste) Kindelement des aktuellen Knotens zurück. Der Startknoten wird der Methode als Eingangsparameter übergeben.

Syntax

```
METHOD ChildByName : SXmlNode
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Rückgabewert

Name	Typ

Eingänge

Name	Typ
n	SXmlNode

Beispielaufruf:

```
xmlChild := fbXml.Child(xmlNode);
```

5.2.1.7.36 ChildByAttribute

Diese Methode dient zur Navigation durch den DOM und liefert eine Referenz auf ein Kindelement im XML-Dokument zurück. Der Startknoten sowie der Name und der Wert des Attributs werden der Methode als Eingangsparameter übergeben.

Syntax

```
METHOD ChildByAttribute : SXmlNode
VAR_INPUT
  n     : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  attr  : STRING;
  value : STRING;
END_VAR
```

Rückgabewert

Name	Typ
ChildByAttribute	SXmlNode

Eingänge

Name	Typ
n	SXmlNode

Ein-/Ausgänge

Name	Typ
attr	STRING
value	STRING

Beispielaufruf:

```
xmlMachine1 := fbXml.ChildByAttribute(xmlMachines, 'Type', '1');
```

5.2.1.7.37 ChildByAttributeAndName

Diese Methode dient zur Navigation durch den DOM und liefert eine Referenz auf ein Kindelement im XML-Dokument zurück. Der Startknoten, der Name und der Wert des Attributs sowie der Name des Kindelements werden der Methode als Eingangsparameter übergeben.

Syntax

```
METHOD ChildByAttributeAndName : SXmlNode
VAR_INPUT
  n     : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  attr  : STRING;
  value : STRING;
  child : STRING;
END_VAR
```

Rückgabewert

Name	Typ
ChildByAttributeAndName	SXmlNode

Eingänge

Name	Typ
n	SXmlNode

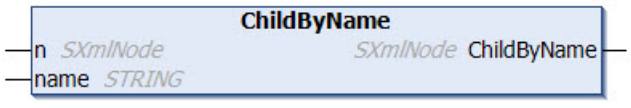
/ Ein-/Ausgänge

Name	Typ
attr	STRING
value	STRING
child	STRING

Beispielaufruf:

```
xmlMachine2 := fbXml.ChildByAttributeAndName(xmlMachines, 'Type', '2', 'Machine');
```

5.2.1.7.38 ChildByName



Diese Methode dient zur Navigation durch den DOM und liefert eine Referenz auf ein Kindelement im XML-Dokument zurück. Der Startknoten und der Name des zurückzuliefernden Elements werden der Methode als Eingangsparameter übergeben.

Syntax

```
METHOD ChildByName : SXmlNode
VAR_INPUT
  n    : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name : STRING;
END_VAR
```

Rückgabewert

Name	Typ
ChildByName	SXmlNode

Eingänge

Name	Typ
n	SXmlNode

/ Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

```
xmlMachines := fbXml.ChildByName(xmlDoc, 'Machines');
```

5.2.1.7.39 Children

Diese Methode dient zur Navigation durch den DOM und liefert einen Iterator auf mehrere gefundene Kindelemente im XML-Dokument zurück. Der Iterator kann dann zur weiteren Navigation durch die gefundenen Elemente verwendet werden. Der Startknoten und eine Referenz auf den Iterator werden der Methode als Eingangsparameter übergeben.

Syntax

```
METHOD Children : SXmlNode
VAR_INPUT
  n  : SXmlNode;
  it : REFERENCE TO SXmlIterator;
END_VAR
```

Rückgabewert

Name	Typ
Children	SXmlNode

Eingänge

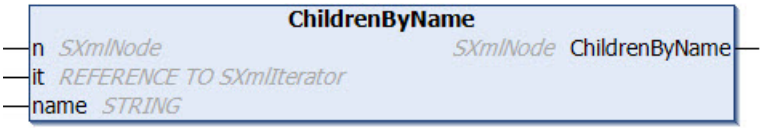
Name	Typ
n	SXmlNode

Beispielaufruf:

```
xmlRet := fbXml.Children(xmlNode, xmlIterator);
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlMachineNodeRef  := fbXml.Node(xmlIterator);
```

```
  xmlMachineNodeText := fbXml.NodeText(xmlMachineNodeRef);
  xmlIterator        := fbXml.Next(xmlIterator);
END_WHILE
```

5.2.1.7.40 ChildrenByName



Diese Methode dient zur Navigation durch den DOM und liefert einen Iterator auf mehrere gefundene Kindelemente im XML-Dokument zurück. Der Iterator kann dann zur weiteren Navigation durch die gefundenen Elemente verwendet werden. Der Startknoten sowie der Name der zu suchenden Kindelemente und eine Referenz auf den Iterator werden der Methode als Eingangsparameter übergeben.

Syntax

```
METHOD ChildrenByName : SXmlNode
VAR_INPUT
  n    : SXmlNode;
  it   : REFERENCE TO SXmlIterator;
END_VAR
VAR_IN_OUT CONSTANT
  name : STRING;
END_VAR
```

Rückgabewert

Name	Typ
ChildrenByName	SXmlNode

Eingänge

Name	Typ
n	SXmlNode
it	REFERENCE TO SXmlIterator

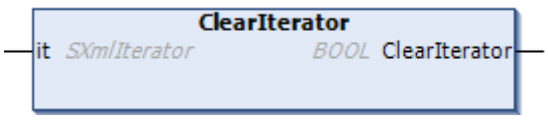
/ Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

```
xmlMachineNode := fbXml.ChildrenByName(xmlMachines, xmlIterator, 'Machine');
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlMachineNodeRef  := fbXml.Node(xmlIterator);
  xmlMachineNodeText := fbXml.NodeText(xmlMachineNodeRef);
  xmlIterator        := fbXml.Next(xmlIterator);
END_WHILE
```

5.2.1.7.41 ClearIterator



Diese Methode setzt bereits verwendete Iteratoren zurück, sodass diese bei dem nächsten Programmdurchlauf wieder verwendet werden können.

Syntax

```
METHOD ClearIterator : BOOL
VAR_INPUT
  it : SXmlIterator;
END_VAR
```

Rückgabewert

Name	Typ
ClearIterator	BOOL

Eingänge

Name	Typ
it	SXmlIterator

Beispielaufruf:

```
bResult := fbXml.ClearIterator(xmlIt);
```

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4024.56	x86, x64, ARM	Tc3_JsonXml 3.4.5.0

5.2.1.7.42 Compare

Diese Methode überprüft zwei Iteratoren auf Gleichheit.

Syntax

```
METHOD Compare : BOOL
VAR_INPUT
  it1 : SXmlIterator;
  it2 : SXmlIterator;
END_VAR
```

Rückgabewert

Name	Typ
Compare	BOOL

Eingänge

Name	Typ
It1	SXmlIterator
It2	SXmlIterator

Beispielaufruf:

```
bResult := fbXml.Compare(xmlIt1, xmlIt2);
```

5.2.1.7.43 CopyAttributeText

Diese Methode liest den Wert eines XML-Attributs aus und schreibt diesen in eine Variable vom Datentyp String. Das XML-Attribut sowie die Zielvariable und die zu schreibende Länge werden der Methode als Eingangsparameter übergeben. Die tatsächliche Größe wird von der Methode zurückgeliefert.

Syntax

```
METHOD CopyAttributeText : UDINT
VAR_INPUT
  a    : SXmlAttribute;
END_VAR
VAR_IN_OUT CONSTANT
  sXml : STRING;
END_VAR
VAR_INPUT
  nXml : UDINT;
END_VAR
```

Rückgabewert

Name	Typ
CopyAttributeText	UDINT

Eingänge

Name	Typ
a	SXmlAttribute
nXml	UDINT

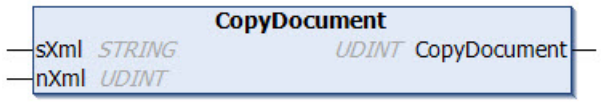
Ein-/Ausgänge

Name	Typ
sXml	STRING

Beispielaufruf:

```
nLength := fbXml.CopyAttributeText(xmlAttr, sTarget, SIZEOF(sTarget));
```

5.2.1.7.44 CopyDocument



Diese Methode kopiert den Inhalt des DOM-Speichers in eine Variable vom Datentyp String. Die zu schreibende Länge und die Variable, in die der resultierende String geschrieben werden soll, werden der Methode als Eingangsparameter übergeben. Die tatsächlich geschriebene Länge wird von der Methode zurückgeliefert. Beachten Sie, dass die Größe der String-Variablen mindestens der Größe des XML-Dokuments im DOM entspricht.

Syntax

```
METHOD CopyDocument : UDINT
VAR_IN_OUT CONSTANT
  sXml : STRING;
END_VAR
```

```
VAR_INPUT
  nXml : UDINT;
END_VAR
```

Rückgabewert

Name	Typ
CopyDocument	UDINT

Eingänge

Name	Typ
nXml	UDINT

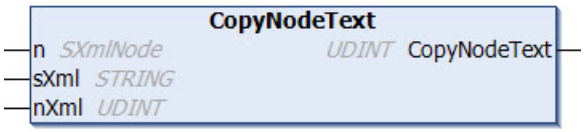
/ Ein-/Ausgänge

Name	Typ
sXml	STRING

Beispielaufruf:

```
nLength := fbXml.CopyDocument(sTarget, SIZEOF(sTarget));
```

5.2.1.7.45 CopyNodeText



Diese Methode liest den Wert eines XML-Knotens aus und schreibt diesen in eine Variable vom Datentyp String. Der XML-Knoten sowie die Zielvariable und die zu schreibende Länge werden der Methode als Eingangsparameter übergeben. Die tatsächliche Größe wird von der Methode zurückgeliefert.

Syntax

```
METHOD CopyNodeText : UDINT
VAR_INPUT
  n    : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  sXml : STRING;
END_VAR
VAR_INPUT
  nXml : UDINT;
END_VAR
```

Rückgabewert

Name	Typ
CopyNodeText	UDINT

Eingänge

Name	Typ
n	SXmlNode
nXml	UDINT

/ Ein-/Ausgänge

Name	Typ
sXml	STRING

Beispielaufruf:

```
nLength := fbXml.CopyNodeText(xmlNode, sTarget, SIZEOF(sTarget));
```

5.2.1.7.46 CopyNodeXml

Diese Methode liest die XML-Struktur eines XML Knotens aus und schreibt diese in eine Variable vom Datentyp String. Der XML-Knoten sowie die Zielvariable und die zu schreibende Länge werden der Methode als Eingangsparameter übergeben. Die tatsächliche Größe wird von der Methode zurückgeliefert.

Syntax

```
METHOD CopyNodeXml : UDINT
VAR_INPUT
  a    : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  sXml : STRING;
END_VAR
VAR_INPUT
  nXml : UDINT;
END_VAR
```

Rückgabewert

Name	Typ
CopyNodeXml	UDINT

Eingänge

Name	Typ
a	SXmlNode
nXml	UDINT

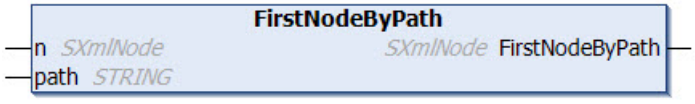
/ Ein-/Ausgänge

Name	Typ
sXml	STRING

Beispielaufruf:

```
nLength := fbXml.CopyNodeXml(xmlNode, sTarget, SIZEOF(sTarget));
```

5.2.1.7.47 FirstNodeByPath



Diese Methode navigiert anhand eines an die Methode übergebenen Pfads durch ein XML-Dokument. Der Pfad und der Startknoten werden der Methode als Eingangsparameter übergeben. Der Pfad wird mit "/" als Separator spezifiziert. Als Rückgabewert liefert die Methode eine Referenz auf den gefundenen XML-Knoten.

Syntax

```
METHOD FirstNodeByPath : SXmlNode
VAR_INPUT
  n    : SXmlNode;
  path : STRING;
END_VAR
```

Rückgabewert

Name	Typ
FirstNodeByPath	SXmlNode

Eingänge

Name	Typ
n	SXmlNode
path	STRING

Beispielaufruf:

```
xmlFoundNode := fbXml.FirstNodeByPath(xmlStartNode, 'Level1/Level2/Level3');
```

5.2.1.7.48 GetAttributeTextLength

Diese Methode liefert die Länge des Werts eines XML-Attributs. Das XML-Attribut wird der Methode als Eingangsparameter übergeben.

Syntax

```
METHOD GetAttributeTextLength : UDINT
VAR_INPUT
  a : SXmlAttribute;
END_VAR
```

Rückgabewert

Name	Typ
GetAttributeTextLength	UDINT

Eingänge

Name	Typ
a	SXmlAttribute

Beispielaufruf:

```
nLength := fbXml.GetAttributeTextLength(xmlAttr);
```

5.2.1.7.49 GetDocumentLength

Diese Methode liefert die Länge eines XML-Dokuments in Bytes.

Syntax

```
METHOD GetDocumentLength : UDINT
```

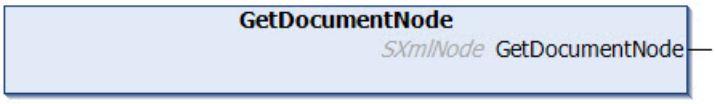
Rückgabewert

Name	Typ
GetDocumentLength	UDINT

Beispielaufruf:

```
nLength := fbXml.GetDocumentLength();
```

5.2.1.7.50 GetDocumentNode



Diese Methode liefert den Root-Knoten eines XML-Dokuments. Dieser ist nicht gleichbedeutend mit dem ersten XML-Knoten im Dokument (verwenden Sie hierfür die Methode GetRootNode()). Die Methode kann auch verwendet werden, um ein leeres XML-Dokument im DOM zu erzeugen.

Syntax

```
METHOD GetDocumentNode : SXmlNode
```

Rückgabewert

Name	Typ
GetDocumentNode	SXmlNode

Beispielaufruf:

```
objRoot := fbXml.GetDocumentNode();
```

5.2.1.7.51 GetNodeTextLength

Diese Methode liefert die Länge des Werts eines XML-Knotens. Der XML-Knoten wird der Methode als Eingangsparameter übergeben.

Syntax

```
METHOD GetNodeTextLength : UDINT
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Rückgabewert

Name	Typ
GetNodeTextLength	UDINT

Eingänge

Name	Typ
n	SXmlNode

Beispielaufruf:

```
nLength := fbXml.GetNodeTextLength(xmlNode);
```

5.2.1.7.52 GetNodeXmlLength

Diese Methode liefert die Länge der XML-Struktur eines XML-Knotens. Der XML-Knoten wird der Methode als Eingangsparameter übergeben.

Syntax

```
METHOD GetNodeXmlLength : UDINT
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Rückgabewert

Name	Typ
GetNodeXmlLength	UDINT

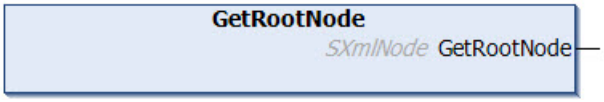
Eingänge

Name	Typ
n	SXmlNode

Beispielaufruf:

```
nLength := fbXml.GetNodeXmlLength(xmlNode);
```

5.2.1.7.53 GetRootNode



Diese Methode liefert eine Referenz zum ersten XML-Knoten im XML-Dokument.

Syntax

```
METHOD GetRootNode : SXmlNode
```

Rückgabewert

Name	Typ
GetRootNode	SXmlNode

Beispielaufruf:

```
xmlRootNode := fbXml.GetRootNode();
```

5.2.1.7.54 InsertAttributeCopy

Diese Methode fügt ein Attribut zu einem XML-Knoten hinzu und kopiert hierbei den Name und den Wert eines existierenden Attributs. Das Attribut kann hierbei an einer bestimmten Stelle platziert werden. Der XML-Knoten, die Position und eine Referenz auf das existierende Attribut-Objekt werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf das neu hinzugefügte Attribut.

Syntax

```
METHOD InsertAttributeCopy : SXmlAttribute
VAR_INPUT
  n      : SXmlNode;
  before : SXmlAttribute;
  copy   : SXmlAttribute;
END_VAR
```

Rückgabewert

Name	Typ
InsertAttributeCopy	SXmlAttribute

Eingänge

Name	Typ
n	SXmlNode
before	SXmlAttribute
copy	SXmlAttribute

Beispielaufruf:

```
xmlNewAttr := fbXml.InsertAttributeCopy(xmlNode, xmlBeforeAttr, xmlCopyAttr);
```

5.2.1.7.55 InsertAttribute

Diese Methode fügt ein Attribut zu einem XML-Knoten hinzu. Das Attribut kann hierbei an einer bestimmten Stelle platziert werden. Der XML-Knoten, die Position und der Name des neuen Attributs werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf das neu hinzugefügte Attribut. Anschließend kann z. B. über die Methode SetAttribute() ein Wert für das Attribut eingetragen werden.

Syntax

```
METHOD InsertAttribute : SXmlAttribute
VAR_INPUT
  n      : SXmlNode;
  before : SXmlAttribute;
END_VAR
VAR_IN_OUT CONSTANT
  name   : STRING;
END_VAR
```

Rückgabewert

Name	Typ
InsertAttribute	SXmlAttribute

Eingänge

Name	Typ
n	SXmlNode
before	SXmlAttribute

/ Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

```
xmlNewAttr := fbXml.InsertAttribute(xmlNode, xmlBeforeAttr, 'SomeName');
```

5.2.1.7.56 InsertChild

Diese Methode fügt einen Knoten zu einem existierenden XML-Knoten hinzu. Der neue Knoten kann hierbei an einer bestimmten Stelle platziert werden. Der existierende XML-Knoten sowie die Position und der Name des neuen Knotens werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf den neu hinzugefügten Knoten. Anschließend kann z. B. über die Methode SetChild() ein Wert für den Knoten eingetragen werden.

Syntax

```
METHOD InsertChild : SXmlNode
VAR_INPUT
  n      : SXmlNode;
  before : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name   : STRING;
END_VAR
```

Rückgabewert

Name	Typ
InsertChild	SXmlNode

Eingänge

Name	Typ
n	SXmlNode
before	SXmlAttribute

/ Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

```
xmlNewNode := fbXml.InsertChild(xmlNode, xmlBeforeNode, 'SomeName');
```

5.2.1.7.57 InsertCopy

Diese Methode fügt einen neuen Knoten zu einem existierenden XML Knoten hinzu und kopiert hierbei einen existierenden Knoten. Der neue Knoten kann hierbei an einer beliebigen Stelle im existierenden Knoten platziert werden. Der XML-Knoten, die Position und eine Referenz auf das existierende Knoten-Objekt werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf den neu hinzugefügten Knoten.

Syntax

```
METHOD InsertCopy : SXmlNode
VAR_INPUT
  n      : SXmlNode;
  before : SXmlNode;
  copy   : SXmlNode;
END_VAR
```

Rückgabewert

Name	Typ
InsertCopy	SXmlNode

Eingänge

Name	Typ
n	SXmlNode
before	SXmlNode
copy	SXmlNode

Beispielaufruf:

```
xmlNewNode := fbXml.InsertCopy(xmlNode, xmlBeforeNode, xmlCopyNode);
```

5.2.1.7.58 IsEnd

Diese Methode überprüft, ob sich ein gegebener XML-Iterator am Ende der zu durchlaufenden Iteration befindet.

Syntax

```
METHOD IsEnd : BOOL
VAR_INPUT
  it : SXmlIterator;
END_VAR
```

Rückgabewert

Name	Typ
IsEnd	BOOL

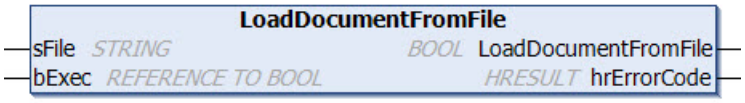
Eingänge

Name	Typ
nit	SXmlIterator

Beispielaufruf:

```
xmlIterator := fbXml.Begin(xmlNode);
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlNodeRef   := fbXml.Node(xmlIterator);
  xmlNodeValue := fbXml.NodeText(xmlNodeRef);
  xmlIterator  := fbXml.Next(xmlIterator);
END_WHILE
```

5.2.1.7.59 LoadDocumentFromFile



Diese Methode lädt ein XML-Dokument aus einer Datei. Der absolute Pfad zu der Datei wird der Methode als Eingangsparameter übergeben.

Eine steigende Flanke am Eingangsparameter bExec triggert hierbei den Ladevorgang. Der asynchrone Prozess ist beendet, sobald die Referenz bExec von der Methode aus zurück auf FALSE gesetzt wird. Mit Beendigung des Prozesses zeigt der Rückgabewert der Methode für einen Aufruf lang an, ob das Laden der Datei erfolgreich (TRUE) oder fehlgeschlagen (FALSE) war.

Syntax

```
METHOD LoadDocumentFromFile : BOOL
VAR_IN_OUT CONSTANT
  sFile      : STRING;
END_VAR
VAR_INPUT
  bExec      : REFERENCE TO BOOL;
END_VAR
VAR_OUTPUT
  hrErrorCode: HRESULT;
END_VAR
```

Rückgabewert

Name	Typ
LoadDocumentFromFile	BOOL

Eingänge

Name	Typ
bExec	REFERENCE TO BOOL

Ein-/Ausgänge

Name	Typ
sFile	STRING

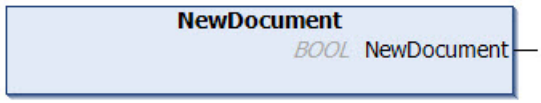
Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

```
IF bLoad THEN
  bLoaded := fbXml.LoadDocumentFromFile('C:\Test.xml', bLoad);
END_IF
```

5.2.1.7.60 NewDocument



Diese Methode erzeugt ein leeres XML-Dokument im DOM-Speicher.

Syntax

```
METHOD NewDocument : BOOL
```

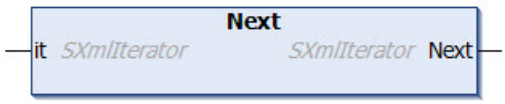
Rückgabewert

Name	Typ
NewDocument	BOOL

Beispielaufruf:

```
fbXml.NewDocument();
```

5.2.1.7.61 Next



Diese Methode setzt einen XML-Iterator auf das nächste zu durchlaufende Objekt.

Syntax

```
METHOD Next : SXmlIterator
VAR_INPUT
  it : SXmlIterator;
END_VAR
```

Rückgabewert

Name	Typ
Next	SXmlIterator

Eingänge

Name	Typ
it	SXmlIterator

Beispielaufruf:

```
xmlIterator := fbXml.Begin(xmlNode);
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlNodeRef   := fbXml.Node(xmlIterator);
  xmlNodeValue := fbXml.NodeText(xmlNodeRef);
  xmlIterator  := fbXml.Next(xmlIterator);
END_WHILE
```

5.2.1.7.62 NextAttribute

Diese Methode liefert zu einem gegebenen XML-Attribut das nächste Attribut.

Syntax

```
METHOD NextAttribute : SXmlAttribute
VAR_INPUT
  a : SXmlAttribute;
END_VAR
```

Rückgabewert

Name	Typ
NextAttribute	SXmlAttribute

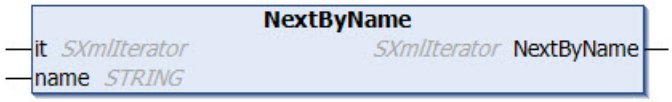
Eingänge

Name	Typ
a	SXmlAttribute

Beispielaufruf:

```
xmlNextAttr := fbXml.NextAttribute(xmlAttr);
```

5.2.1.7.63 NextByName



Diese Methode setzt einen XML-Iterator auf das nächste zu durchlaufende Objekt, das anhand seines Namens identifiziert wird.

Syntax

```
METHOD NextByName : SXmlIterator
VAR_INPUT
  it   : SXmlIterator;
END_VAR
VAR_IN_OUT CONSTANT
  name : STRING;
END_VAR
```

Rückgabewert

Name	Typ
NextByName	SXmlIterator

Eingänge

Name	Typ
it	SXmlIterator

/ Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

```
xmlIterator := fbXml.Begin(xmlNode);
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlNodeRef   := fbXml.Node(xmlIterator);
  xmlNodeValue := fbXml.NodeText(xmlNodeRef);
  xmlIterator  := fbXml.NextByName(xmlIterator, 'SomeName');
END_WHILE
```

5.2.1.7.64 NextSibling

Diese Methode liefert zu einem gegebenen XML-Knoten den nächsten direkten Knoten auf derselben XML-Ebene.

Syntax

```
METHOD NextSibling : SXmlNode
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Rückgabewert

Name	Typ
NextSibling	SXmlNode

Eingänge

Name	Typ
n	SXmlNode

Beispielaufruf:

```
xmlSibling := fbXml.NextSibling(xmlNode);
```

5.2.1.7.65 NextSiblingByName

Diese Methode liefert zu einem gegebenen XML-Knoten den nächsten direkten Knoten mit einem bestimmten Namen auf derselben XML-Ebene.

Syntax

```
METHOD NextSiblingByName : SXmlNode
VAR_INPUT
  n    : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name : STRING;
END_VAR
```

Rückgabewert

Name	Typ
NextSiblingByName	SXmlNode

Eingänge

Name	Typ
n	SXmlNode

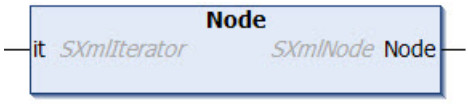
/ Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

```
xmlSibling := fbXml.NextSiblingByName(xmlNode, 'SomeName');
```

5.2.1.7.66 Node



Diese Methode wird in Zusammenhang mit einem Iterator verwendet, um durch den DOM zu navigieren. Der Iterator wird der Methode als Eingangsparameter übergeben. Als Rückgabewert liefert die Methode dann den aktuellen XML-Knoten.

Syntax

```
METHOD Node : SXmlNode
VAR_INPUT
  it : SXmlIterator;
END_VAR
```

Rückgabewert

Name	Typ
Node	SXmlNode

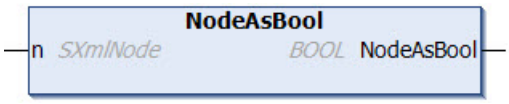
Eingänge

Name	Typ
it	SXmlIterator

Beispielaufruf:

```
xmlMachineNode := fbXml.ChildrenByName(xmlMachines, xmlIterator, 'Machine');
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlMachineNode      := fbXml.Node(xmlIterator);
  xmlMachineNodeValue := fbXml.NodeText(xmlMachineNode);
  xmlIterator         := fbXml.Next(xmlIterator);
END_WHILE
```

5.2.1.7.67 NodeAsBool



Diese Methode liefert den Text eines XML-Knotens als Datentyp Boolean. Der XML-Knoten wird der Methode als Eingangsparameter übergeben.

Syntax

```
METHOD NodeAsBool : BOOL
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Rückgabewert

Name	Typ
NodeAsBool	BOOL

Eingänge

Name	Typ
n	SXmlNode

Beispielaufruf:

```
bXmlNode:= fbXml.NodeAsBool(xmlMachine1);
```

5.2.1.7.68 NodeAsDouble

Diese Methode liefert den Text eines XML-Knotens als Datentyp Double. Der XML-Knoten wird der Methode als Eingangsparameter übergeben.

Syntax

```
METHOD NodeAsDouble : LREAL
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Rückgabewert

Name	Typ
NodeAsDouble	LREAL

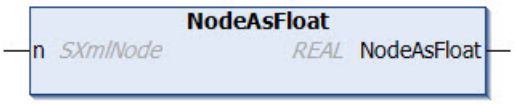
Eingänge

Name	Typ
n	SXmlNode

Beispielaufruf:

```
lrXmlNode:= fbXml.NodeAsDouble(xmlMachine1);
```

5.2.1.7.69 NodeAsFloat



Diese Methode liefert den Text eines XML-Knotens als Datentyp Float. Der XML-Knoten wird der Methode als Eingangsparameter übergeben.

Syntax

```
METHOD NodeAsFloat : REAL
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Rückgabewert

Name	Typ
NodeAsFloat	REAL

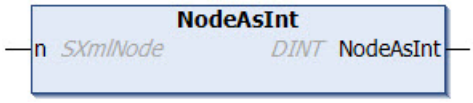
Eingänge

Name	Typ
n	SXmlNode

Beispielaufruf:

```
rXmlNode:= fbXml.NodeAsFloat(xmlMachine1);
```

5.2.1.7.70 NodeAsInt



Diese Methode liefert den Text eines XML-Knotens als Datentyp Integer. Der XML-Knoten wird der Methode als Eingangsparameter übergeben.

Syntax

```
METHOD NodeAsInt : DINT
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Rückgabewert

Name	Typ
NodeAsInt	DINT

Eingänge

Name	Typ
n	SXmlNode

Beispielaufruf:

```
nXmlNode:= fbXml.NodeAsInt(xmlMachine1);
```

5.2.1.7.71 NodeAsLint

Diese Methode liefert den Text eines XML-Knotens als Datentyp Integer64. Der XML-Knoten wird der Methode als Eingangsparameter übergeben.

Syntax

```
METHOD NodeAsLint : LINT
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Rückgabewert

Name	Typ
NodeAsLint	LINT

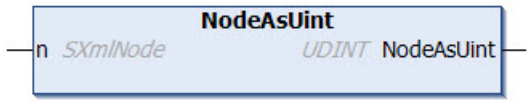
Eingänge

Name	Typ
n	SXmlNode

Beispielaufruf:

```
nXmlNode:= fbXml.NodeAsLint(xmlMachine1);
```

5.2.1.7.72 NodeAsUint



Diese Methode liefert den Text eines XML-Knotens als Datentyp Unsigned Integer. Der XML-Knoten wird der Methode als Eingangsparameter übergeben.

Syntax

```
METHOD NodeAsUint : UDINT
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Rückgabewert

Name	Typ
NodeAsUint	UDINT

Eingänge

Name	Typ
n	SXmlNode

Beispielaufruf:

```
nXmlNode:= fbXml.NodeAsUint(xmlMachine1);
```

5.2.1.7.73 NodeAsUlint

Diese Methode liefert den Text eines XML-Knotens als Datentyp Unsigned Integer64. Der XML-Knoten wird der Methode als Eingangsparameter übergeben.

Syntax

```
METHOD NodeAsUlint : ULINT
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Rückgabewert

Name	Typ
NodeAsUlint	ULINT

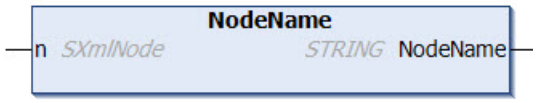
Eingänge

Name	Typ
n	SXmlNode

Beispielaufruf:

```
nXmlNode:= fbXml.NodeAsUlint(xmlMachine1);
```

5.2.1.7.74 NodeName



Diese Methode gibt den Namen eines XML-Knotens zurück. Eine Referenz zum XML-Knoten wird der Methode als Eingangsparameter übergeben.

Syntax

```
METHOD NodeName : STRING
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Rückgabewert

Name	Typ
NodeName	STRING

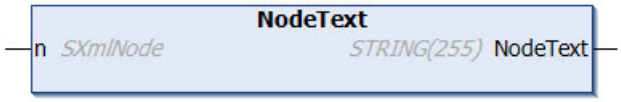
Eingänge

Name	Typ
n	SXmlNode

Beispielaufruf:

```
sNodeName := fbXml.NodeName(xmlMachine1);
```

5.2.1.7.75 NodeText



Diese Methode liefert den Text eines XML-Knotens. Der XML-Knoten wird der Methode als Eingangsparameter übergeben.

Wenn der Text eines XML-Knotens länger als 255 Zeichen ist, muss die Methode CopyNodeText [▶ 282] verwendet werden.

Syntax

```
METHOD NodeText : STRING(255)
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Rückgabewert

Name	Typ
NodeText	STRING(255)

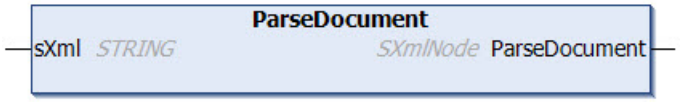
Eingänge

Name	Typ
n	SXmlNode

Beispielaufruf:

```
sMachine1Name := fbXml.NodeText(xmlMachine1);
```

5.2.1.7.76 ParseDocument



Diese Methode lädt ein XML-Dokument zur weiteren Verarbeitung in den DOM-Speicher. Das XML-Dokument liegt hierbei als String vor und wird der Methode als Eingangsparameter übergeben. Eine Referenz zum XML-Dokument im DOM wird an den Aufrufer zurückgegeben.

Syntax

```
METHOD ParseDocument : SXmlNode
VAR_IN_OUT CONSTANT
  sXml : STRING;
END_VAR
```

Rückgabewert

Name	Typ
ParseDocument	SXmlNode

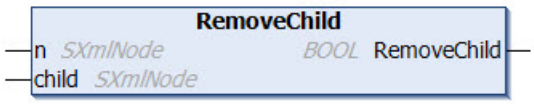
Ein-/Ausgänge

Name	Typ
sXml	STRING

Beispielaufruf:

```
xmlDoc := fbXml.ParseDocument(sXmlToParse);
```

5.2.1.7.77 RemoveChild



Diese Methode entfernt einen XML-Kindknoten von einem gegebenen XML-Knoten. Die beiden XML-Knoten werden der Methode als Eingangsparameter übergeben. Die Methode gibt TRUE zurück, wenn der Vorgang erfolgreich war und der XML-Knoten entfernt wurde.

Syntax

```
METHOD RemoveChild : BOOL
VAR_INPUT
  n     : SXmlNode;
  child : SXmlNode;
END_VAR
```

Rückgabewert

Name	Typ
RemoveChild	BOOL

Eingänge

Name	Typ
n	SXmlNode
child	SXmlNode

Beispielaufruf:

```
bRemoved := fbXml.RemoveChild(xmlParent, xmlChild);
```

5.2.1.7.78 RemoveChildByName

Diese Methode entfernt einen XML-Kindknoten von einem gegebenen XML-Knoten. Der zu entfernende Knoten wird über dessen Namen angesprochen. Gibt es mehr als einen Kindknoten, so wird der letzte entfernt. Die Methode gibt TRUE zurück, wenn der Vorgang erfolgreich war und der XML-Knoten entfernt wurde.

Syntax

```
METHOD RemoveChildByName : BOOL
VAR_INPUT
  n     : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name : STRING;
END_VAR
```

Rückgabewert

Name	Typ
RemoveChildByName	BOOL

Eingänge

Name	Typ
n	SXmlNode

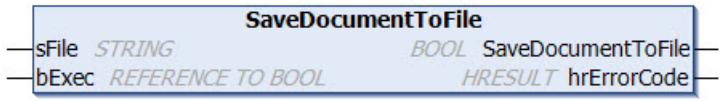
/ Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

```
bRemoved := fbXml.RemoveChildByName(xmlParent, 'SomeName');
```

5.2.1.7.79 SaveDocumentToFile



Diese Methode speichert das aktuelle XML-Dokument in einer Datei. Der absolute Pfad zu der Datei wird der Methode als Eingangsparameter übergeben.

Eine steigende Flanke am Eingangsparameter bExec triggert hierbei den Speichervorgang. Der asynchrone Prozess ist beendet, sobald die Referenz bExec von der Methode aus zurück auf FALSE gesetzt wird. Mit Beendigung des Prozesses zeigt der Rückgabewert der Methode für einen Aufruf lang an, ob das Speichern der Datei erfolgreich (TRUE) oder fehlgeschlagen (FALSE) war.

Syntax

```
METHOD SaveDocumentToFile : BOOL
VAR_IN_OUT CONSTANT
  sFile      : STRING;
END_VAR
VAR_INPUT
  bExec      : REFERENCE TO BOOL;
END_VAR
VAR_OUTPUT
  hrErrorCode: HRESULT;
END_VAR
```

Rückgabewert

Name	Typ
SaveDocumentToFile	BOOL

Eingänge

Name	Typ
bExec	REFERENCE TO BOOL

/ Ein-/Ausgänge

Name	Typ
sFile	STRING

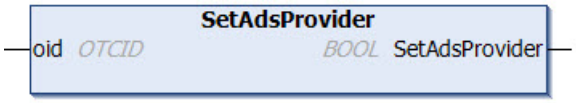
Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

```
IF bSave THEN
  bSaved = fbXml.SaveDocumentToFile('C:\Test.xml', bSave);
END_IF
```

5.2.1.7.80 SetAdsProvider



Syntax

```
METHOD SetAdsProvider : BOOL
VAR_IN_OUT CONSTANT
  oid : OTCID;
END_VAR
```

Rückgabewert

Name	Typ
SetAdsProvider	BOOL

/ Ein-/Ausgänge

Name	Typ
oid	OTCID

5.2.1.7.81 SetAttribute

Diese Methode setzt den Wert eines Attributs. Der Wert ist hierbei vom Datentyp String.

Syntax

```
METHOD SetAttribute : SXmlAttribute
VAR_INPUT
  a     : SXmlAttribute;
END_VAR
VAR_IN_OUT CONSTANT
  value : STRING;
END_VAR
```

Rückgabewert

Name	Typ
SetAttribute	SXmlAttribute

Eingänge

Name	Typ
a	SXmlAttribute

/ Ein-/Ausgänge

Name	Typ
value	STRING

Beispielaufruf:

```
xmlAttr := fbXml.SetAttribute(xmlExistingAttr, 'Test');
```

5.2.1.7.82 SetAttributeAsBool

Diese Methode setzt den Wert eines Attributs. Der Wert ist hierbei vom Datentyp Boolean.

Syntax

```
METHOD SetAttributeAsBool : SXmlAttribute
VAR_INPUT
  a     : SXmlAttribute;
  value : BOOL;
END_VAR
```

Rückgabewert

Name	Typ
SetAttributeAsBool	SXmlAttribute

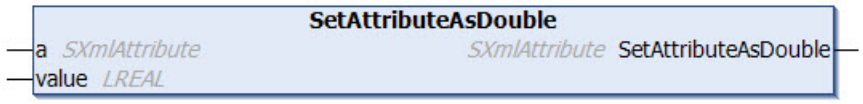
Eingänge

Name	Typ
a	SXmlAttribute
value	BOOL

Beispielaufruf:

```
xmlAttr := fbXml.SetAttributeAsBool(xmlExistingAttr, TRUE);
```

5.2.1.7.83 SetAttributeAsDouble



Diese Methode setzt den Wert eines Attributs. Der Wert ist hierbei vom Datentyp Double.

Syntax

```
METHOD SetAttributeAsDouble : SXmlAttribute
VAR_INPUT
  a     : SXmlAttribute;
  value : LREAL;
END_VAR
```

Rückgabewert

Name	Typ
SetAttributeAsDouble	SXmlAttribute

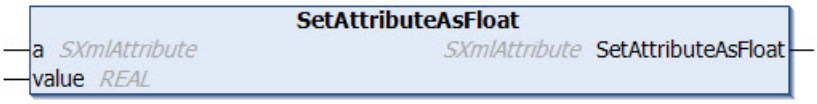
Eingänge

Name	Typ
a	SXmlAttribute
value	LREAL

Beispielaufruf:

```
xmlAttr := fbXml.SetAttributeAsDouble(xmlExistingAttr, 42.42);
```

5.2.1.7.84 SetAttributeAsFloat



Diese Methode setzt den Wert eines Attributs. Der Wert ist hierbei vom Datentyp Float.

Syntax

```
METHOD SetAttributeAsFloat : SXmlAttribute
VAR_INPUT
  a     : SXmlAttribute;
  value : REAL;
END_VAR
```

Rückgabewert

Name	Typ
SetAttributeAsFloat	SXmlAttribute

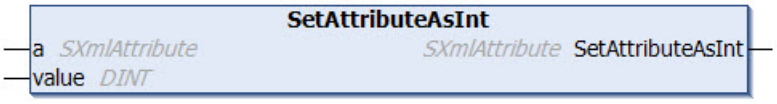
Eingänge

Name	Typ
a	SXmlAttribute
value	REAL

Beispielaufruf:

```
xmlAttr := fbXml.SetAttributeAsFloat(xmlExistingAttr, 42.42);
```

5.2.1.7.85 SetAttributeAsInt



Diese Methode setzt den Wert eines Attributs. Der Wert ist hierbei vom Datentyp Integer.

Syntax

```
METHOD SetAttributeAsInt : SXmlAttribute
VAR_INPUT
  a     : SXmlAttribute;
  value : DINT;
END_VAR
```

Rückgabewert

Name	Typ
SetAttributeAsInt	SXmlAttribute

Eingänge

Name	Typ
a	SXmlAttribute
value	DINT

Beispielaufruf:

```
xmlAttr := fbXml.SetAttributeAsInt(xmlExistingAttr, 42);
```

5.2.1.7.86 SetAttributeAsLint

Diese Methode setzt den Wert eines Attributs. Der Wert ist hierbei vom Datentyp Integer64.

Syntax

```
METHOD SetAttributeAsLint : SXmlAttribute
VAR_INPUT
  a     : SXmlAttribute;
  value : LINT;
END_VAR
```

Rückgabewert

Name	Typ
SetAttributeAsLint	SXmlAttribute

Eingänge

Name	Typ
a	SXmlAttribute
value	LINT

Beispielaufruf:

```
xmlAttr := fbXml.SetAttributeAsLint(xmlExistingAttr, 42);
```

5.2.1.7.87 SetAttributeAsUint

Diese Methode setzt den Wert eines Attributs. Der Wert ist hierbei vom Datentyp Unsigned Integer.

Syntax

```
METHOD SetAttributeAsUint : SXmlAttribute
VAR_INPUT
  a     : SXmlAttribute;
  value : UDINT;
END_VAR
```

Rückgabewert

Name	Typ
SetAttributeAsUint	SXmlAttribute

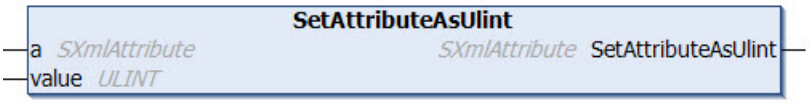
Eingänge

Name	Typ
a	SXmlAttribute
value	UDINT

Beispielaufruf:

```
xmlAttr := fbXml.SetAttributeAsUint(xmlExistingAttr, 42);
```

5.2.1.7.88 SetAttributeAsUlint



Diese Methode setzt den Wert eines Attributs. Der Wert ist hierbei vom Datentyp Unsigned Integer64.

Syntax

```
METHOD SetAttributeAsUlint : SXmlAttribute
VAR_INPUT
  a     : SXmlAttribute;
  value : ULINT;
END_VAR
```

Rückgabewert

Name	Typ
SetAttributeAsUlint	SXmlAttribute

Eingänge

Name	Typ
a	SXmlAttribute
value	ULINT

Beispielaufruf:

```
xmlAttr := fbXml.SetAttributeAsUlint(xmlExistingAttr, 42);
```

5.2.1.7.89 SetChild

Diese Methode setzt den Wert eines XML-Knotens. Der Wert wird der Methode als Eingangsparameter vom Datentyp String übergeben. Der Eingangsparameter cdata gibt an, ob der Wert des Knotens in einem CDATA-Block gekapselt werden soll, sodass bestimmte Sonderzeichen wie z. B. "<" und ">" als Werte erlaubt sind.

Syntax

```
METHOD SetChild : SXmlNode
VAR_INPUT
  n     : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  value : STRING;
END_VAR
```

```
VAR_INPUT
  cdata : BOOL;
END_VAR
```

Rückgabewert

Name	Typ
SetChild	SXmlNode

Eingänge

Name	Typ
n	SXmlNode
cdata	BOOL

/ Ein-/Ausgänge

Name	Typ
value	STRING

Beispielaufruf:

```
xmlNode := fbXml.SetChild(xmlExistingNode, 'SomeText', FALSE);
```

5.2.1.7.90 SetChildAsBool

Diese Methode setzt den Wert eines XML-Knotens. Der Wert wird der Methode als Eingangsparameter vom Datentyp Boolean übergeben.

Syntax

```
METHOD SetChildAsBool : SXmlNode
VAR_INPUT
  n     : SXmlNode;
  value : BOOL;
END_VAR
```

Rückgabewert

Name	Typ
SetChildAsBool	SXmlNode

Eingänge

Name	Typ
n	SXmlNode
value	BOOL

Beispielaufruf:

```
xmlNode := fbXml.SetChild(xmlExistingNode, TRUE);
```

5.2.1.7.91 SetChildAsDouble

Diese Methode setzt den Wert eines XML-Knotens. Der Wert wird der Methode als Eingangsparameter vom Datentyp Double übergeben.

Syntax

```
METHOD SetChildAsDouble : SXmlNode
VAR_INPUT
  n     : SXmlNode;
  value : LREAL;
END_VAR
```

Rückgabewert

Name	Typ
SetChildAsDouble	SXmlNode

Eingänge

Name	Typ
n	SXmlNode
value	LREAL

Beispielaufruf:

```
xmlNode := fbXml.SetChildAsDouble(xmlExistingNode, 42.42);
```

5.2.1.7.92 SetChildAsFloat

Diese Methode setzt den Wert eines XML-Knotens. Der Wert wird der Methode als Eingangsparameter vom Datentyp Float übergeben.

Syntax

```
METHOD SetChildAsFloat : SXmlNode
VAR_INPUT
  n     : SXmlNode;
  value : REAL;
END_VAR
```

Rückgabewert

Name	Typ
SetChildAsFloat	SXmlNode

Eingänge

Name	Typ
n	SXmlNode
value	REAL

Beispielaufruf:

```
xmlNode := fbXml.SetChildAsFloat(xmlExistingNode, 42.42);
```

5.2.1.7.93 SetChildAsInt

Diese Methode setzt den Wert eines XML-Knotens. Der Wert wird der Methode als Eingangsparameter vom Datentyp Integer übergeben.

Syntax

```
METHOD SetChildAsInt : SXmlNode
VAR_INPUT
  n     : SXmlNode;
  value : DINT;
END_VAR
```

Rückgabewert

Name	Typ
SetChildAsInt	SXmlNode

Eingänge

Name	Typ
n	SXmlNode
value	DINT

Beispielaufruf:

```
xmlNode := fbXml.SetChildAsInt(xmlExistingNode, 42);
```

5.2.1.7.94 SetChildAsLint

Diese Methode setzt den Wert eines XML-Knotens. Der Wert wird der Methode als Eingangsparameter vom Datentyp Integer64 übergeben.

Syntax

```
METHOD SetChildAsLint : SXmlNode
VAR_INPUT
  n     : SXmlNode;
  value : LINT;
END_VAR
```

Rückgabewert

Name	Typ
SetChildAsLint	SXmlNode

Eingänge

Name	Typ
n	SXmlNode
value	LINT

Beispielaufruf:

```
xmlNode := fbXml.SetChildAsLint(xmlExistingNode, 42);
```

5.2.1.7.95 SetChildAsUint

Diese Methode setzt den Wert eines XML-Knotens. Der Wert wird der Methode als Eingangsparameter vom Datentyp Unsigned Integer übergeben.

Syntax

```
METHOD SetChildAsUint : SXmlNode
VAR_INPUT
  n     : SXmlNode;
  value : UDINT;
END_VAR
```

Rückgabewert

Name	Typ
SetChildAsUint	SXmlNode

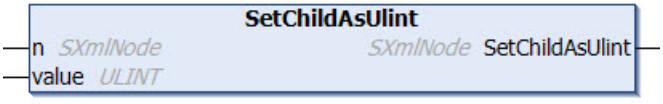
Eingänge

Name	Typ
n	SXmlNode
value	UDINT

Beispielaufruf:

```
xmlNode := fbXml.SetChildAsUint(xmlExistingNode, 42);
```

5.2.1.7.96 SetChildAsUlint



Diese Methode setzt den Wert eines XML-Knotens. Der Wert wird der Methode als Eingangsparameter vom Datentyp Unsigned Integer64 übergeben.

Syntax

```
METHOD SetChildAsUlint : SXmlNode
VAR_INPUT
  n     : SXmlNode;
  value : ULINT;
END_VAR
```

Rückgabewert

Name	Typ
SetChildAsUlint	SXmlNode

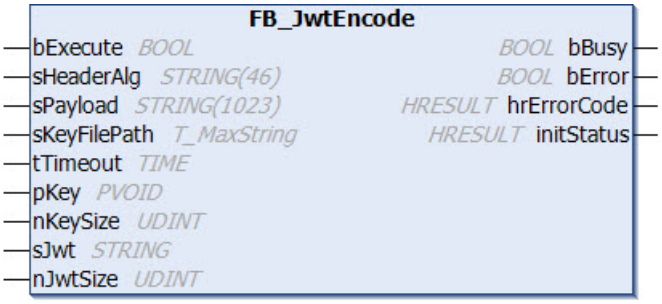
Eingänge

Name	Typ
n	SXmlNode
value	ULINT

Beispielaufruf:

```
xmlNode := fbXml.SetChildAsUlint(xmlExistingNode, 42);
```

5.2.1.8 FB_JwtEncode



Der Funktionsbaustein ermöglicht die Erzeugung und Signierung eines JSON Web Token (JWT).

Syntax

Definition:

```
FUNCTION_BLOCK FB_JwtEncode
VAR_INPUT
  bExecute       : BOOL;
  sHeaderAlg     : STRING(46);
  sPayload       : STRING(1023);
  sKeyFilePath   : STRING(511);
  tTimeout       : TIME;
  pKey           : PVOID;
  nKeySize       : UDINT;
  nJwtSize       : UDINT;
END_VAR
VAR_IN_OUT CONSTANT
  sJwt           : STRING;
END_VAR
VAR_OUTPUT
  bBusy          : BOOL;
  bError         : BOOL;
  hrErrorCode    : HRESULT;
  initStatus     : HRESULT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
bExecute	BOOL	Steigende Flanke aktiviert die Abarbeitung des Funktionsbausteins.
sHeaderAlg	STRING(46)	Der zu verwendende Algorithmus für den JWT Header, z. B. RS256.
sPayload	STRING(1023	Der zu verwendende Payload des JWT.
sKeyFilePath	STRING(511)	Pfad zum Private Key, welcher für die Signatur des JWT verwendet werden soll.
tTimeout	TIME	ADS Timeout, welcher intern für den Dateizugriff auf den Private Key verwendet wird.
pKey	PVOID	Buffer für den auszulesenden Private Key.
nKeySize	UDINT	Maximalgröße des Buffers.
sJwt	STRING [▶ 65]	Enthält nach Abarbeitung des Funktionsbausteins das fertig kodierte und signierte JWT.
nJwtSize	UDINT [▶ 65]	Größe des erzeugten JWT inklusive Nullterminierung.

Ausgänge

Name	Typ	Beschreibung
bBusy	BOOL	Ist TRUE, solange die Abarbeitung des Funktionsbausteins noch nicht beendet ist.
bError	BOOL	Wird TRUE, sobald eine Fehlersituation eintritt.
hrErrorCode	HRESULT	Liefert bei einem gesetzten Ausgang `bError` einen Fehlercode. Eine Erläuterung zu den möglichen Fehlercodes [▶ 347] befindet sich im Anhang.
initStatus	HRESULT	Liefert im Fall einer fehlgeschlagenen Initialisierung des Funktionsbausteins einen Fehlercode.

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4024.4	x86, x64, ARM	Tc3_JsonXml 3.3.6.0

5.2.2 Schnittstellen

5.2.2.1 ITcJsonSaxHandler

5.2.2.1.1 OnBool

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein Wert vom Datentyp BOOL gefunden wurde. Der Eingangsparameter value enthält hierbei den gefundenen Wert. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnBool : HRESULT
VAR_INPUT
    value : BOOL;
END_VAR
```

5.2.2.1.2 OnDint

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein Wert vom Datentyp DINT gefunden wurde. Der Eingangsparameter value enthält hierbei den gefundenen Wert. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnDint : HRESULT
VAR_INPUT
    value : DINT;
END_VAR
```

5.2.2.1.3 OnEndArray

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers eine eckige schließende Klammer gefunden wurde, was dem JSON-Synonym für ein endendes Array entspricht. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnEndArray : HRESULT
```

5.2.2.1.4 OnEndObject

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers eine geschweifte schließende Klammer gefunden wurde, was dem JSON-Synonym für ein endendes Objekt entspricht. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnEndObject : HRESULT
```

5.2.2.1.5 OnKey

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein Property gefunden wurde. Der Property-Name liegt hierbei am Eingangs-/Ausgangsparameter key an und dessen Länge am Eingangsparameter len. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnKey : HRESULT
VAR_IN_OUT CONSTANT
    key : STRING;
END_VAR
VAR_INPUT
    len : UDINT;
END_VAR
```

5.2.2.1.6 OnLint

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein Wert vom Datentyp LINT gefunden wurde. Der Eingangsparameter value enthält hierbei den gefundenen Wert. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnLint : HRESULT
VAR_INPUT
    value : LINT;
END_VAR
```

5.2.2.1.7 OnLreal

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein Wert vom Datentyp LREAL gefunden wurde. Der Eingangsparameter value enthält hierbei den gefundenen Wert. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnLreal : HRESULT
VAR_INPUT
    value : LREAL;
END_VAR
```

5.2.2.1.8 OnNull

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein NULL-Wert gefunden wurde. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnNull : HRESULT
```

5.2.2.1.9 OnStartArray

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers eine eckige öffnende Klammer gefunden wurde, was dem JSON-Synonym für ein beginnendes Array entspricht. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnStartArray : HRESULT
```

5.2.2.1.10 OnStartObject

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers eine geschweifte öffnende Klammer gefunden wurde, was dem JSON-Synonym für ein beginnendes Objekt entspricht. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnStartObject : HRESULT
```

5.2.2.1.11 OnString

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein Wert vom Datentyp STRING gefunden wurde. Der In/Out-Parameter value enthält hierbei den gefundenen Wert. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnString : HRESULT
VAR_IN_OUT CONSTANT
    value : STRING;
END_VAR
VAR_INPUT
    len   : UDINT;
END_VAR
```

5.2.2.1.12 OnUdint

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein Wert vom Datentyp UDINT gefunden wurde. Der Eingangsparameter value enthält hierbei den gefundenen Wert. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnUdint : HRESULT
VAR_INPUT
    value : UDINT;
END_VAR
```

5.2.2.1.13 OnUlint

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein Wert vom Datentyp ULINT gefunden wurde. Der Eingangsparameter value enthält hierbei den gefundenen Wert. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnUlint : HRESULT
VAR_INPUT
    value : ULINT;
END_VAR
```

5.2.2.2 ITcJsonSaxValues

5.2.2.2.1 OnBoolValue

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein Wert vom Datentyp BOOL gefunden wurde. Der Eingangsparameter value enthält hierbei den gefundenen Wert. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnBoolValue : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
    value : BOOL;
END_VAR
```

5.2.2.2.2 OnDintValue

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein Wert vom Datentyp DINT gefunden wurde. Der Eingangsparameter value enthält hierbei den gefundenen Wert. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnDintValue : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
    value : DINT;
END_VAR
```

5.2.2.2.3 OnLintValue

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein Wert vom Datentyp LINT gefunden wurde. Der Eingangsparameter value enthält hierbei den gefundenen Wert. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnLintValue : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
    value : LINT;
END_VAR
```

5.2.2.2.4 OnLrealValue

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein Wert vom Datentyp LREAL gefunden wurde. Der Eingangsparameter value enthält hierbei den gefundenen Wert. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnLrealValue : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
    value : LREAL;
END_VAR
```

5.2.2.2.5 OnNullValue

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein NULL-Wert gefunden wurde. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnNull : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
END_VAR
```

5.2.2.2.6 OnStringValue

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein Wert vom Datentyp STRING gefunden wurde. Der Eingangs-/Ausgangsparameter value enthält hierbei den gefundenen Wert. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnStringValue : HRESULT
VAR_IN_OUT CONSTANT
    value : STRING;
END_VAR
VAR_INPUT
    len : UDINT;
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
END_VAR
```

5.2.2.2.7 OnUdintValue

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein Wert vom Datentyp UDINT gefunden wurde. Der Eingangsparameter value enthält hierbei den gefundenen Wert. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnUdintValue : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
    value : UDINT;
END_VAR
```

5.2.2.2.8 OnUlintValue

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein Wert vom Datentyp ULINT gefunden wurde. Der Eingangsparameter value enthält hierbei den gefundenen Wert. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnUlintValue : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
    value : ULINT;
END_VAR
```

6 Beispiele

Folgende Beispiele behandeln die Kommunikation zu einem MQTT Broker. Es werden Nachrichten verschickt und empfangen.

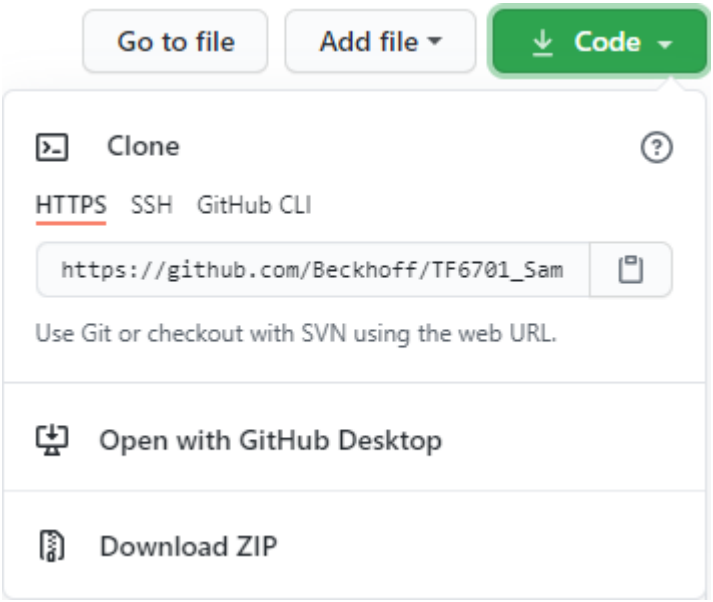
Es gibt zwei unterschiedliche Implementierungsvarianten, zwischen denen nach rein subjektiven Gesichtspunkten gewählt werden kann. Die zwei Möglichkeiten sind mit den ersten beiden Beispielen gegenüber gestellt.

Übersicht

Beispiel	Link	Beschreibung
1	IotMqttSampleUsingQueue [▶ 319]	MQTT-Kommunikation mithilfe einer Nachrichten-Warteschlange
2	IotMqttSampleUsingCallback [▶ 321]	MQTT-Kommunikation über eine Callback-Methode
3	IotMqttSampleTlsPsk [▶ 323]	MQTT-Kommunikation über eine gesicherte TLS-Verbindung mittels PSK (PreSharedKey)
4	IotMqttSampleTlsCa [▶ 324]	MQTT-Kommunikation über eine gesicherte TLS-Verbindung mittels CA(Certificate Authority)-Zertifikat
5	IotMqttSampleAwsIoT [▶ 325]	MQTT-Kommunikation mit AWS IoT
6	IotMqttSampleAzureIotHub [▶ 327]	MQTT-Kommunikation mit dem Microsoft Azure IoT Hub
7	IotMqttSampleIbmWatsonIoT [▶ 330]	MQTT-Kommunikation mit IBM Watson IoT
8	IotMqttSampleMathworksThingspeak [▶ 331]	MQTT-Kommunikation mit der ThingSpeak IoT Plattform von MathWorks
9	IotMqttv5Sample [▶ 333]	MQTTv5-Kommunikation mithilfe einer Nachrichten-Warteschlange
10	IotMqttv5LastWillSample [▶ 336]	Demonstriert die Verwendung von LastWill in Zusammenspiel mit den MQTTv5 Funktionsbausteinen.
11	IotMqttv5ReqResSample [▶ 336]	Demonstriert den Request/Response Mechanismus von MQTTv5, sowie die Handhabung von UserProperties und CorrelationData
12	IotMqttv5UserPropsSample [▶ 337]	Demonstriert die Verwendung von User Properties, welche Bestandteil von MQTTv5 sind.
13	JsonXmlSamples [▶ 338]	Beinhaltet diverse Samples, welche die Verwendung der JSON/XML Parser aus der SPS-Bibliothek Tc3_JsonXml demonstrieren.

Downloads

Beispielcode und -konfigurationen für dieses Produkt können über das entsprechende Repository auf GitHub bezogen werden: https://www.github.com/Beckhoff/TF6701_Samples. Sie haben dort die Möglichkeit das Repository zu clonen oder ein ZIP File mit dem Sample herunterzuladen.



6.1 IotMqttSampleUsingQueue

Beispiel zur MQTT-Kommunikation mithilfe einer Message-Queue

In diesem Beispiel wird die Kommunikation zu einem MQTT Broker dargestellt. Es werden Nachrichten verschickt („Publish“) und empfangen. Dies erfolgt in zwei Schritten. Zuerst wird entschieden, welche Nachrichten generell empfangen werden sollen („Subscribe“). Anschließend werden empfangene Nachrichten in einer Message Queue eingesammelt, um von hier ausgelesen und ausgewertet werden zu können.

Projektstruktur

1. Erstellen Sie ein TwinCAT-Projekt mit einer SPS und fügen Sie die Tc3_IotBase als Bibliotheksreferenz an.
2. Legen Sie einen Programmbaustein an und deklarieren Sie eine Instanz von FB_IotMqttClient [▶ 57] sowie zwei Hilfsvariablen, um den Programmablauf bei Bedarf steuern zu können.

```
PROGRAM PrgMqttCom
VAR
    fbMqttClient    : FB_IotMqttClient;
    bSetParameter   : BOOL := TRUE;
    bConnect        : BOOL := TRUE;
END_VAR
```

3. Deklarieren Sie für die zu verschickende MQTT-Nachricht zwei Variablen für Topic und Payload. Im Beispiel soll jede Sekunde eine Nachricht verschickt werden.

```
    (* published message *)
    sTopicPub   : STRING(255) := 'MyTopic';
    sPayloadPub : STRING(255);
    i : UDINT;
    fbTimer : TON := (PT:=T#1S);
```

4. Deklarieren Sie für den Nachrichtenempfang eine Variable, die das zu empfangene Topic enthält und zwei weitere Variablen, die Topic und Payload der zuletzt empfangenen Nachricht anzeigen.
   Die empfangenen Nachrichten sollen in einer Warteschlange (Queue) gesammelt werden, um eine nach der anderen ausgewertet werden zu können. Hierfür deklarieren Sie eine Instanz von FB_IotMqttMessageQueue [▶ 66] sowie eine Instanz von FB_IotMqttMessage [▶ 68].

```
    (* received message *)
    bSubscribed    : BOOL;
    sTopicSub      : STRING(255) := 'MyTopic';
    {attribute 'TcEncoding':='UTF-8'}
    sTopicRcv      : STRING(255);
    {attribute 'TcEncoding':='UTF-8'}
    sPayloadRcv    : STRING(255);
    fbMessageQueue : FB_IotMqttMessageQueue;
    fbMessage      : FB_IotMqttMessage;
```

5. Im Programmteil muss der MQTT Client zyklisch getriggert werden, um den Verbindungsaufbau zum Broker, den Verbindungserhalt und den Nachrichtenempfang zu gewährleisten. Setzen Sie die Parameter der gewünschten Verbindung und initialisieren Sie den Verbindungsaufbau mit dem Übergabeparameter `bConnect := TRUE`.
   Im Beispiel werden die Parameter einmalig vor dem Client-Aufruf im Programmcode zugewiesen. Weil dies meist nur einmalig nötig ist, können die Parameter auch bereits im Deklarationsteil bei der Instanziierung des MQTT Client angegeben werden. Nicht alle Parameter müssen zugewiesen werden. Im Beispiel ist der Broker lokal. Sie können auch die IP-Adresse oder den Namen angeben.

```
IF bSetParameter THEN
    bSetParameter               := FALSE;
    fbMqttClient.sHostName      := 'localhost';
    fbMqttClient.nHostPort      := 1883;
//  fbMqttClient.sClientId      := 'MyTcMqttClient';
    fbMqttClient.sTopicPrefix   := '';
//  fbMqttClient.nKeepAlive     := 60;
//  fbMqttClient.sUserName      := ;
//  fbMqttClient.sUserPassword  := ;
//  fbMqttClient.stWill         := ;
//  fbMqttClient.stTLS          := ;
    fbMqttClient.ipMessageQueue := fbMessageQueue;
END_IF

fbMqttClient.Execute(bConnect);
```

6. Sobald die Verbindung zum Broker aufgebaut wird, soll sich der Client auf ein bestimmtes Topic anmelden. Ebenso soll jede Sekunde eine Nachricht versandt werden.
   Im Beispiel ist `sTopicPub = sTopicSub`, sodass ein Loopback entsteht. In anderen Applikationen unterscheiden sich die Topics typischerweise.

```
IF fbMqttClient.bConnected THEN
    IF NOT bSubscribed THEN
        bSubscribed := fbMqttClient.Subscribe(sTopic:=sTopicSub, eQoS:=TcIotMqttQos.AtMostOnceDe
livery);
    END_IF
    fbTimer(IN:=TRUE);
    IF fbTimer.Q THEN // publish new payload every second
        fbTimer(IN:=FALSE);
        i := i + 1;
        sPayloadPub := CONCAT('MyMessage', TO_STRING(i));
        fbMqttClient.Publish(    sTopic:= sTopicPub,
                                 pPayload:= ADR(sPayloadPub), nPayloadSize:= LEN2(ADR(sPayloadPub
))+1,
                                 eQoS:= TcIotMqttQos.AtMostOnceDelivery, bRetain:= FALSE, bQueue:
= FALSE );
    END_IF
END_IF
```

7. Der zyklische Aufruf vom MQTT Client sorgt für den Empfang der Nachrichten. Der Client empfängt alle Nachrichten, auf deren Topic er sich zuvor beim Broker angemeldet hat und legt diese in der Message Queue ab. Sobald Nachrichten verfügbar sind, rufen Sie die Methode Dequeue() auf und erhalten über das Nachrichtenobjekt fbMessage Zugriff auf die Nachrichteneigenschaften wie Topic und Payload.

```
IF fbMessageQueue.nQueuedMessages > 0 THEN
    IF fbMessageQueue.Dequeue(fbMessage:=fbMessage) THEN
        fbMessage.GetTopic(pTopic:=ADR(sTopicRcv), nTopicSize:=SIZEOF(sTopicRcv) );
        fbMessage.GetPayload(pPayload:=ADR(sPayloadRcv), nPayloadSize:=SIZEOF(sPayloadRcv), bSet
NullTermination:=FALSE);
    END_IF
END_IF
```

Bei obiger Implementierung der Nachrichtenauswertung wird pro Zyklus eine empfangene Nachricht ausgewertet. Falls mehrere Nachrichten in der Message-Queue gesammelt wurden, verteilt sich deren Auswertung entsprechend auf mehrere Zyklen.

Das Beispiel kann abgewandelt werden für Applikationen, in denen sich auf mehrere Topics anmeldet wird. Dann werden MQTT-Nachrichten mit verschiedenen Topics empfangen. Die Auswertung der Nachrichten könnten Sie wie folgt für diesen Anwendungsfall erweitern:

```
VAR
    (* received payload for each subscribed topic *)
    sPayloadRcv1 : STRING(255);
    sPayloadRcv2 : STRING(255);
END_VAR
VAR CONSTANT
    (* subscriptions *)
    sTopicSub1 : STRING(255) := 'my first topic';
```

```
    sTopicSub2 : STRING(255) := 'my second topic';
END_VAR
-----------------------------------------------------------------

IF fbMessageQueue.nQueuedMessages > 0 THEN
    IF fbMessageQueue.Dequeue(fbMessage:=fbMessage) THEN
        IF fbMessage.CompareTopic(sTopic:=sTopicSub1) THEN
            fbMessage.GetPayload(pPayload:=ADR(sPayloadRcv1), nPayloadSize:=SIZEOF(sPayloadRcv1), bS
etNullTermination:=FALSE);
        ELSIF fbMessage.CompareTopic(sTopic:=sTopicSub2) THEN
            fbMessage.GetPayload(pPayload:=ADR(sPayloadRcv2), nPayloadSize:=SIZEOF(sPayloadRcv2), bS
etNullTermination:=FALSE);
        END_IF
    END_IF
END_IF
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.0	IPC oder CX (x86, x64, ARM)	Tc3_IotBase, Tc2_Utilities (>= v3.3.19.0)

6.2 IotMqttSampleUsingCallback

Beispiel zur MQTT Kommunikation über eine Callback-Methode

In diesem Beispiel wird die Kommunikation zu einem MQTT Broker dargestellt. Es werden Nachrichten verschickt („Publish“) und empfangen. Dies erfolgt in zwei Schritten. Zuerst wird entschieden, welche Nachrichten generell empfangen werden sollen („Subscribe“). Anschließend werden, während dem zyklischen Aufruf der FB_IotMqttClient.Execute()-Methode, neue Nachrichten über eine Callback-Methode empfangen.

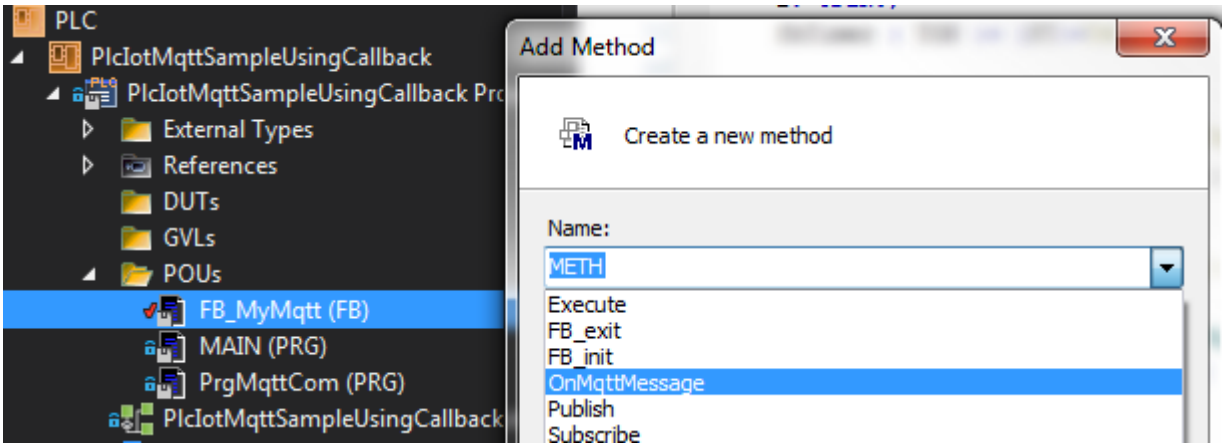
Projektstruktur

1. Erstellen Sie ein TwinCAT-Projekt mit einer SPS und fügen Sie die Tc3_IotBase als Bibliotheksreferenz an.
2. Die Callback-Methode, in der die empfangenen MQTT-Nachrichten bereitgestellt werden, möchten Sie selbst implementieren. Damit der TwinCAT-Treiber diese Methode aufrufen kann, wird das Prinzip der Vererbung genutzt. Legen Sie zunächst einen Funktionsbaustein an und lassen Sie diesen von dem Funktionsbaustein FB_IotMqttClient erben. In diesem Funktionsbaustein können Sie bereits Teile der MQTT-Kommunikation kapseln. Im Beispiel werden hier empfangene Nachrichten ausgewertet. Deklarieren Sie aus diesem Grund Variablen für Topic und Payload.

```
{attribute 'c++_compatible'}
FUNCTION_BLOCK FB_MyMqtt EXTENDS FB_IotMqttClient

VAR
    (* received message *)
    {attribute 'TcEncoding':='UTF-8'}
    sTopicRcv   : STRING(255);
    {attribute 'TcEncoding':='UTF-8'}
    sPayloadRcv : STRING(255);
END_VAR
```

3. Legen Sie die Methode OnMqttMessage() an und überschreiben Sie die Basisimplementierung.



4. Die Methode mit der von Ihnen vorzunehmenden Implementierung wird nicht in der Applikation aufgerufen, sondern implizit vom Treiber. Dieser Callback findet während des zyklischen Triggerns des Clients statt und kann je nach Anzahl der seit dem letzten Trigger empfangenen Nachrichten keinmal, einmal oder mehrfach erfolgen. Wie folgendes Code Snippet zeigt, implementieren Sie in diesem Beispiel nur eine einfache Auswertung.

```
{attribute 'c++_compatible'}
{attribute 'pack_mode' := '4'}
{attribute 'show'}
{attribute 'minimal_input_size' := '4'}
METHOD OnMqttMessage : HRESULT
VAR_IN_OUT CONSTANT
    topic    : STRING;
END_VAR
VAR_INPUT
    payload  : PVOID;
    length   : UDINT;
    qos      : TcIotMqttQos;
    repeated : BOOL;
END_VAR
VAR
    nPayloadRcvLen : UDINT;
END_VAR
-------------------------------------------------------------------------

SUPER^.nMessagesRcv := SUPER^.nMessagesRcv + 1;

STRNCPY( ADR(sTopicRcv), ADR(topic), SIZEOF(sTopicRcv) );
nPayloadRcvLen := MIN(length, DINT_TO_UDINT(SIZEOF(sPayloadRcv))-1);
MEMCPY( ADR(sPayloadRcv), payload, nPayloadRcvLen );
sPayloadRcv[nPayloadRcvLen] := 0;  // ensure a null termination of received string

OnMqttMessage := S_OK;
```

5. Die weiteren Schritte sind ähnlich zum Beispiel MQTT Kommunikation mithilfe einer Message-Queue [▶ 319].
   Legen Sie einen Programmbaustein an und deklarieren Sie eine Instanz von Ihrem zuvor definierten Funktionsbaustein FB_MyMqtt sowie zwei Hilfsvariablen, um den Programmablauf bei Bedarf steuern zu können.

```
PROGRAM PrgMqttCom
VAR
    fbMqttClient  : FB_MyMqtt;
    bSetParameter : BOOL := TRUE;
    bConnect      : BOOL := TRUE;
END_VAR
```

6. Deklarieren Sie für die zu verschickende MQTT-Nachricht zwei Variablen für Topic und Payload. Im Beispiel soll jede Sekunde eine Nachricht verschickt werden.

```
    (* published message *)
    sTopicPub : STRING(255) := 'MyTopic';
    sPayloadPub : STRING(255);
    i: UDINT;
    fbTimer : TON := (PT:=T#1S);
```

7. Deklarieren Sie für den Nachrichtenempfang eine Variable, die das zu empfangene Topic enthält.

```
    bSubscribed : BOOL;
    sTopicSub   : STRING(255) := 'MyTopic';
```

8. Im Programmteil muss der MQTT Client zyklisch getriggert werden, um den Verbindungsaufbau zum Broker, den Verbindungserhalt und den Nachrichtenempfang zu gewährleisten. Setzen Sie die Parameter der gewünschten Verbindung und initialisieren Sie den Verbindungsaufbau mit dem Übergabeparameter `bConnect := TRUE`. Im Beispiel werden die Parameter einmalig vor dem Client-Aufruf im Programmcode zugewiesen. Weil dies meist nur einmalig nötig ist, können die Parameter auch bereits im Deklarationsteil bei der Instanziierung des MQTT Client angegeben werden. Nicht alle Parameter müssen zugewiesen werden. Im Beispiel ist der Broker lokal. Sie können auch die IP-Adresse oder den Namen angeben.

```
IF bSetParameter THEN
    bSetParameter              := FALSE;
    fbMqttClient.sHostName     := 'localhost';
    fbMqttClient.nHostPort     := 1883;
//  fbMqttClient.sClientId     := 'MyTcMqttClient';
    fbMqttClient.sTopicPrefix  := '';
//  fbMqttClient.nKeepAlive    := 60;
//  fbMqttClient.sUserName     := ;
//  fbMqttClient.sUserPassword := ;
//  fbMqttClient.stWill        := ;
//  fbMqttClient.stTLS         := ;
END_IF

fbMqttClient.Execute(bConnect);
```

9. Sobald die Verbindung zum Broker aufgebaut wird, soll sich der Client auf ein bestimmtes Topic anmelden. Ebenso soll jede Sekunde eine Nachricht versandt werden.
Im Beispiel ist `sTopicPub = sTopicSub`, so dass ein Loopback entsteht. In anderen Applikationen unterscheiden sich die Topics typischerweise.

```
IF fbMqttClient.bConnected THEN
    IF NOT bSubscribed THEN
        bSubscribed := fbMqttClient.Subscribe(sTopic:=sTopicSub, eQoS:=TcIotMqttQos.AtMostOnceDe
livery);
    END_IF
    fbTimer(IN:=TRUE);
    IF fbTimer.Q THEN // publish new payload every second
        fbTimer(IN:=FALSE);
        i := i + 1;
        sPayloadPub := CONCAT('MyMessage', TO_STRING(i));
        fbMqttClient.Publish(    sTopic:= sTopicPub,
                                 pPayload:= ADR(sPayloadPub), nPayloadSize:= LEN2(ADR(sPayloadPub
))+1,
                                 eQoS:= TcIotMqttQos.AtMostOnceDelivery, bRetain:= FALSE, bQueue:
= FALSE );
    END_IF
END_IF
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.0	IPC oder CX (x86, x64, ARM)	Tc3_IotBase, Tc2_Utilities (>= v3.3.19.0)

6.3 IotMqttSampleTlsPsk

Beispiel zur MQTT Kommunikation über eine gesicherte TLS Verbindung mittels PSK (PreSharedKey)

In diesem Beispiel wird die Kommunikation zu einem MQTT Broker dargestellt, welcher eine Authentifizierung via TLS-PSK verlangt. Das Beispiel beschränkt sich im Wesentlichen auf das Herstellen der Verbindung und den Publish von Werten.

Projektstruktur

1. Erstellen Sie ein TwinCAT-Projekt mit einer SPS und fügen Sie die Tc3_IotBase als Bibliotheksreferenz hinzu.
2. Legen Sie einen Programmbaustein an und deklarieren Sie eine Instanz von FB_IotMqttClient [▶ 57], sowie zwei Hilfsvariablen, um den Programmablauf bei Bedarf steuern zu können.

```
PROGRAM PrgMqttCom
VAR
  fbMqttClient    : FB_IotMqttClient;
  bSetParameter   : BOOL := TRUE;
  bConnect : BOOL := TRUE;
END_VAR
```

3. Deklarieren Sie für die zu verschickende MQTT Nachricht zwei Variablen für Topic und Payload. Im Beispiel soll jede Sekunde eine Nachricht verschickt werden.

```
sTopicPub   : STRING(255) := 'MyTopic';
sPayloadPub : STRING(255);
i : UDINT;
fbTimer : TON := (PT:=T#1S);
```

4. Im Programmteil muss der MQTT Client zyklisch getriggert werden, um den Verbindungsaufbau zum Broker und den Verbindungserhalt zu gewährleisten. Setzen Sie die Parameter der gewünschten Verbindung und initialisieren Sie den Verbindungsaufbau mit dem Übergabeparameter `bConnect := TRUE`. Im Beispiel werden die Parameter einmalig vor dem Client-Aufruf im Programmcode zugewiesen. Weil dies meist nur einmalig nötig ist, können die Parameter auch bereits im Deklarationsteil bei der Instanziierung des MQTT Client angegeben werden. Nicht alle Parameter müssen zugewiesen werden. Im Beispiel ist der Broker lokal. Sie können auch die IP-Adresse oder den Namen angeben

```
IF bSetParameter THEN
  bSetParameter                  := FALSE;
  fbMqttClient.stTLS.sPskIdentity := 'my_Identity';
  fbMqttClient.stTLS.aPskKey      := cMyPskKey;
  fbMqttClient.stTLS.nPskKeyLen   := 15;
  fbMqttClient.nHostPort          := 8883;
END_IF

fbMqttClient.Execute(bConnect);
```

5. Das Strukturelement aPskKey bekommt den PreSharedKey übergeben, welcher für den Verbindungsaufbau zum Broker benötigt wird. Dieser muss entsprechend als ein ARRAY OF BYTE der Länge 64 angegeben werden. Die tatsächliche Länge des Keys wird dann dem Strukturelement nPskKeyLen übergeben.
6. Sobald die Verbindung zum Broker aufgebaut wird, soll der Client jede Sekunde eine Nachricht an ein bestimmtes Topic versenden.

```
IF fbMqttClient.bConnected THEN
  fbTimer(IN:=TRUE);
  IF fbTimer.Q THEN // publish new payload every second
    fbTimer(IN:=FALSE);
    i := i + 1;
    sPayloadPub := CONCAT('MyMessage', TO_STRING(i));
    fbMqttClient.Publish(sTopic:= sTopicPub,
                         pPayload:= ADR(sPayloadPub),
                         nPayloadSize:= LEN2(ADR(sPayloadPub))+1,
                         eQoS:= TcIotMqttQos.AtMostOnceDelivery,
                         bRetain:= FALSE,
                         bQueue:= FALSE);
  END_IF
END_IF
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.0	IPC oder CX (x86, x64, ARM)	Tc3_IotBase, Tc2_Utilities (>= v3.3.19.0)

6.4 IotMqttSampleTlsCa

Beispiel zur MQTT Kommunikation über eine gesicherte TLS-Verbindung mittels CA

In diesem Beispiel wird die Kommunikation zu einem MQTT Broker dargestellt, welcher eine Authentifizierung via TLS und Client-Zertifikat verlangt. Dieses Beispiel ist nicht als separater Download erhältlich, da es im Wesentlichen auf den bereits existierenden Beispielen IotMqttSampleUsingQueue

[▶ 319] und vor allem auch IotMqttSampleAwsIoT [▶ 325] basiert. Letzteres demonstriert bereits die Verwendung von Client-Zertifikaten mit TF6701 und kann in identischer Art und Weise auch für alle anderen MQTT Broker verwendet werden.

Parameter für Verbindungsaufbau

Das folgende Code Snippet zeigt die notwendigen Parameter für einen TLS-Verbindungsaufbau mit einem MQTT Broker via Client-Zertifikat. Im Wesentlichen handelt es sich hierbei um statische Parameter. Diese können auch im Deklarationsteil bei der Instanziierung des MQTT Client angegeben werden.

```
IF bSetParameter THEN
  bSetParameter := FALSE;
  fbMqttClient.stTLS.sCA := 'c:\TwinCAT\3.1\Config\Certificates\rootCa.pem';
  fbMqttClient.stTLS.sCert := 'c:\TwinCAT\3.1\Config\Certificates\clientCert.pem.crt';
  fbMqttClient.stTLS.sKeyFile := 'c:\TwinCAT\3.1\Config\Certificates\clientPrivKey.pem.key';
END_IF
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.0	IPC oder CX (x86, x64, ARM)	Tc3_IotBase, Tc2_Utilities (>= v3.3.19.0)

6.5 IotMqttSampleAwsIoT

Beispiel zur MQTT Kommunikation mit AWS IoT Core

In diesem Beispiel wird die Kommunikation zum AWS IoT Core Message Broker gezeigt, welcher Bestandteil der AWS IoT Plattform ist. Der Message Broker benötigt eine Authentifizierung via TLS Client-Zertifikat. Als Voraussetzung hierfür, muss das entsprechende Zertifikat erstellt und auf der AWS IoT Plattform bekannt und aktiviert worden sein. Sie können hierbei auf der AWS IoT Management Console selbst-signierte Zertifikate verwenden, oder auch Zertifikate von einer eigenen Certificate Authority (CA) signieren lassen. In letztem Fall muss der eigenen CA entsprechend von der AWS IoT Core Plattform vertraut werden.

Ersteinrichtung von AWS IoT Core

Informationen zur Erstellung und Registrierung von Client-Zertifikaten und der Ersteinrichtung von AWS IoT Core finden Sie in der offiziellen AWS IoT Core Dokumentation. Das dort erstellte und aktivierte Zertifikat wird von den MQTT Funktionsbausteinen verwendet, um eine Verbindung mit dem Message Broker herzustellen. Beachten Sie auch, dass Sie eine gültige AWS IoT Policy mit dem erstellten Zertifikat verknüpft haben. Wir empfehlen auch folgende weiterführende Artikel in der AWS IoT Core Dokumentation:

AWS IoT Core Security and Identity

X.509 Certificates Authentication

Topic-Struktur

Die Topic-Struktur ist beim AWS IoT Core Message Broker mit Einschränkungen frei wählbar. Es gibt bestimmte System-Topics, welche nicht verwendet werden dürfen. Nähere Informationen finden Sie in der AWS IoT Core Dokumentation. Wir empfehlen auch die AWS-Dokumentation zu AWS IoT Core MQTT topic design.

QoS und Retain

AWS IoT Core unterstützt aktuell kein QoS 2, sowie keine Retain-Nachrichten. Um persistente Nachrichten zu speichern, müssen weitere Dienste, wie z. B. AWS IoT Device Shadow oder auch ein Datenbankservice, verwendet werden.

AWS IoT Core Service Limits

Bitte beachten Sie bei der Verwendung von AWS IoT Core auch die Hinweise zu den AWS Service Limits.

In diesem Beispiel werden sowohl Nachrichten an den AWS IoT Core Message Broker versendet als auch von dort empfangen. Da dieses Beispiel im Wesentlichen auf dem Beispiel IotMqttSampleUsingQueue [▶ 319] basiert, werden in diesem Abschnitt nur die relevanten Teile zur Verbindungsherstellung mit AWS IoT Core erklärt.

Parameter für Verbindungsaufbau

Das folgende Code Snippet zeigt die notwendigen Parameter für einen Verbindungsaufbau mit AWS IoT Core. Im Wesentlichen handelt es sich hierbei um statische Parameter. Diese können auch im Deklarationsteil bei der Instanziierung des MQTT Client angegeben werden.

Für den Verbindungsaufbau mit AWS IoT Core werden folgende Informationen benötigt.

Parameter	Beschreibung
fbMqttClient.stTLS.sCA	Amazon Root CA Zertifikat, welches von der AWS IoT Management Konsole heruntergeladen werden kann.
fbMqttClient.stTLS.sCert	Zertifikat des Geräts, welches zur Authentifizierung für die Verbindungsherstellung zu AWS IoT benötigt wird. Hierbei können sowohl selbst-signierte Zertifikate als auch CA-signierte Zertifikate verwendet werden. Im Falle von CA-signierten Zertifikaten muss die eigene CA auf der AWS IoT Core Management Konsole als vertrauenswürdig eingestuft werden. Weitere Informationen hierzu entnehmen Sie bitte der AWS IoT Core Dokumentation.
fbMqttClient.stTLS.sKeyFile	Private Key des Geräts.
fbMqttClient.sHostname	Hostname der AWS-IoT-Broker-Instanz
fbMqttClient.nHostPort	Da eine Verbindung mit dem AWS IoT Broker nur via TLS hergestellt werden kann, muss hier der Default MQTT TLS Port verwendet werden (8883).
fbMqttClient.sClientId	Die MQTT ClientID kann dem Thing-Namen entsprechen.
Exponential backoff	Die im Code Snippet gezeigte Verwendung eines Exponential Backoff Algorithmus ist optional, wird jedoch empfohlen, siehe unten.

```
IF bSetParameter THEN
  bSetParameter := FALSE;
  fbMqttClient.stTLS.sCA := 'c:\TwinCAT\3.1\Config\Certificates\AmazonRootCA1.pem';
  fbMqttClient.stTLS.sCert := 'c:\TwinCAT\3.1\Config\Certificates\7613eee18a-certificate.pem.crt';
  fbMqttClient.stTLS.sKeyFile := 'c:\TwinCAT\3.1\Config\Certificates\7613eee18a-private.pem.key';
  fbMqttClient.sHostName:= 'a35raby201xp77.iot.eu-west-1.amazonaws.com';
  fbMqttClient.nHostPort:= 8883;
  fbMqttClient.sClientId:= 'CX-12345';
  fbMqttClient.ipMessageQueue := fbMessageQueue;
  fbMqttClient.ActivateExponentialBackoff(T#1S, T#30S);
END_IF
```

Exponential backoff

Um im Falle eines Verbindungsfehlers mit dem Message Broker diesen nicht unnötig mit Verbindungsanforderungen zu belasten, kann auf eine Funktionalität namens "exponential backoff" zurückgegriffen werden. Hierbei wird nach einem TLS-Verbindungsfehler mit dem Message Broker die Reconnect-Rate multiplikativ angepasst. Diese Funktion ist über die Methode ActivateExponentialBackoff() [▶ 63] aktivierbar. Die Parameter der Methode geben hierbei die Minimal- und Maximalzeit für den Algorithmus an. Die Minimalzeit beschreibt den Anfangsverzögerungswert für den erneuten Verbindungsversuch. Die Maximalzeit beschreibt den größten Verzögerungswert. Die Verzögerungswerte werden bis zum Erreichen des Maximalwerts verdoppelt. Sobald eine Verbindung hergestellt werden konnte, wird die Backoff-Rate wieder auf den Ursprungswert zurückgesetzt. Über die Methode DeactivateExponentialBackoff() [▶ 64] kann diese Funktion auch programmatisch wieder außer Kraft gesetzt werden.

Device Shadow Service

Der AWS IoT Device Shadow Service ermöglicht die persistente Speicherung von Zustandsinformationen eines verbundenen Geräts. Für jedes Gerät wird hierbei ein eigener Shadow verwaltet. Der Shadow eines Geräts ist über MQTT auslese- und aktualisierbar. Hierfür müssen bestimmte System-Topics vom AWS IoT Core verwendet werden. Das folgende Topic ermöglicht zum Beispiel die Aktualisierung des Shadows.

```
sTopicShadowUpdate : STRING(255) := '$$aws/things/CX-12345/shadow/update';
```

Die Nachricht die an dieses Topic gesendet wird beschreibt den neuen Shadow des Geräts. Sie wird in JSON Notation angegeben und entspricht hierbei einem bestimmten Format, z. B.:

```
{
  "state": {
    "reported": {
      "Vendor": "Beckhoff Automation",
      "CpuTemperature": 42,
      "OperatingSystem": "Windows 10"
    }
  }
}
```

Mit Hilfe der SPS Bibliothek Tc3_JsonXml kann dieses Format erstellt und entsprechend der eigenen Applikation angepasst werden.

In dem hier bereitgestellten SPS-Code wird der Shadow des Geräts einmal beim Start des Beispiels und dann auf Anforderung (abhängig von der Variablen bUpdateShadow) aktualisiert.

Für weitere Informationen zum AWS IoT Device Shadow Service, der dort verwendeten Topics und Datenformate, konsultieren Sie die AWS IoT Device Shadow Service Dokumentation.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.0	IPC oder CX (x86, x64, ARM)	Tc3_IotBase, Tc2_Utilities (>= v3.3.19.0)

6.6 IotMqttSampleAzureIotHub

Beispiel zur MQTT Kommunikation mit Microsoft Azure IoT Hub

In diesem Beispiel wird die Kommunikation zum Microsoft Azure IoT Hub gezeigt, welcher Bestandteil der Microsoft Azure Cloud ist. Der Message Broker ist über MQTT erreichbar und benötigt zur Authentifizierung ein sogenanntes SAS-Token, welches sich über die Azure IoT Hub Plattform generieren lässt, z. B. mithilfe des sogenannten Azure IoT Explorers.

Ersteinrichtung vom Azure IoT Hub

Für Informationen zur Ersteinrichtung vom Microsoft Azure IoT Hub und entsprechender Zugangsdaten für zu verbindende Geräte konsultieren Sie die offizielle Microsoft Azure IoT Hub Dokumentation. Wir empfehlen auch den Microsoft Dokumentationsartikel zur Verwendung von MQTT mit dem Azure IoT Hub.

Topic-Struktur

Die Topic-Struktur beim Senden und Empfangen von Nachrichten ist fest vom Microsoft Azure IoT Hub vorgegeben.

Authentifizierung

Zur Authentifizierung des MQTT Clients können Sie entweder ein SAS Token oder X509 Zertifikate verwenden.

QoS und Retain

Der Azure IoT Hub unterstützt kein QoS 2 und keine Retain-Nachrichten.

In diesem Beispiel werden sowohl Nachrichten an den Azure IoT Hub versendet als auch empfangen. Da das Beispiel im Wesentlichen somit auf dem Beispiel IotMqttSampleUsingQueue [▶ 319] basiert, werden in diesem Abschnitt nur die relevanten Teile zur Verbindungsherstellung mit dem IoT Hub erklärt.

Parameter für Verbindungsaufbau

Das folgende Code Snippet zeigt die notwendigen Parameter für einen Verbindungsaufbau mit dem Azure IoT Hub. Im Wesentlichen handelt es sich hierbei um statische Parameter. Diese können auch im Deklarationsteil bei der Instanziierung des MQTT Client angegeben werden.

```
IF bSetParameter THEN
  bSetParameter := FALSE;

  (* Option 1: authentication via Device SAS Token *)
  fbMqttClient.stTLS.sCA := 'c:
\TwinCAT\3.1\Config\Certificates\DigiCertGlobalRootG2.cer'; // CA certificate
  fbMqttClient.stTLS.sAzureSas := 'PlaceDeviceSasTokenHere'; // Device SAS Token

  (* Option 2: authentication via X509 certificate *)
  //fbMqttClient.stTLS.sCA := 'c:
\TwinCAT\3.1\Config\Certificates\DigiCertGlobalRootG2.cer'; // CA certificate
  //fbMqttClient.stTLS.sCert := 'c:\TwinCAT\3.1\Config\Certificates\MyDeviceCert.pem';
  //fbMqttClient.stTLS.sKeyFile := 'c:\TwinCAT\3.1\Config\Certificates\MyDeviceCert.key';
  //fbMqttClient.sHostName:= myIotHub.azure-devices.net';
  //fbMqttClient.nHostPort:= 8883;
  //fbMqttClient.sClientId := 'MyDevice';
  //fbMqttClient.sUserName := 'myIotHub.azure-devices.net/MyDevice/?api-version=2021-04-12';
   fbMqttClient.ipMessageQueue := fbMessageQueue;
END_IF
```

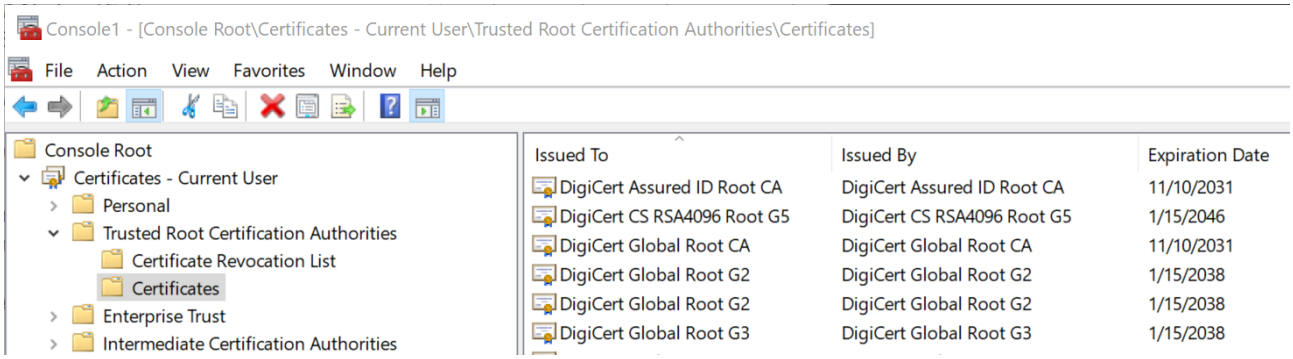
Für die Authentifizierung des MQTT Clients am Azure IoT Hub können Sie entweder ein SAS Token oder X509 Zertifikate verwenden. Je nach Authentifizierungstyp müssen bestimmte Verbindungsparameter gesetzt werden. Die folgende Tabelle gibt einen Überblick über die zu setzenden Parameter.

	SAS Token	X509 Zertifikat
stTLS.sCA	Pfad zum vom Azure IoT Hub verwendeten CA Zertifikat. Siehe unten.	Pfad zum vom Azure IoT Hub verwendeten CA Zertifikat. Siehe unten.
stTLS.sAzureSas	SAS Token, welches über den Azure File Explorer generiert werden kann.	---
stTLS.sCert	---	Pfad zum Client-Zertifikat.
stTLS.sKey	---	Pfad zum Private Key.
sHostname	Muss nicht explizit gesetzt werden. Der Hostname ergibt sich aus dem SAS Token.	Hostname der Azure IoT Hub Instanz.
nHostPort	Muss nicht explizit gesetzt werden. Der Port wird bei Verwendung eines SAS Token automatisch auf 8883 eingestellt.	8883
sClientId	Muss nicht explizit gesetzt werden. Die Client-ID ergibt sich aus dem SAS Token.	Entspricht dem Device-Name des erzeugten Geräts im Azure IoT Hub.
sUserName	Muss nicht explizit gesetzt werden. Der Username ergibt sich aus dem SAS Token.	Entspricht einem festgelegten Namensschema, welches sowohl den IoT Hub Namen als auch Device-Name beinhaltet. Weitere Informationen hierzu entnehmen Sie bitte der Azure IoT Hub Dokumentation.

CA-Zertifikat

Bei Herstellung einer Verbindung zum Microsoft Azure IoT Hub über MQTT ist die Angabe eines CA-Zertifikats zwingend erforderlich, sowohl bei Verwendung eines SAS Token als auch bei Verwendung von X509 Zertifikaten zur Client-Authentifizierung. Bitte konsultieren Sie immer auch die Microsoft Dokumentation, um die aktuell gültige Root-CA zu erfahren.

In den meisten Fällen kann der zugehörige Public Key der jeweiligen Root-CA aus der Microsoft Windows Zertifikatskonsole extrahiert werden (**Start** > **Ausführen** > **mmc.exe**, dann das **SnapIn** „Zertifikate“ hinzufügen). Die Root-CA befindet sich dann unter der Rubrik **Vertrauenswürdige Stammzertifizierungsstellen.** Siehe auch unten stehenden Screenshot als Vergleich.

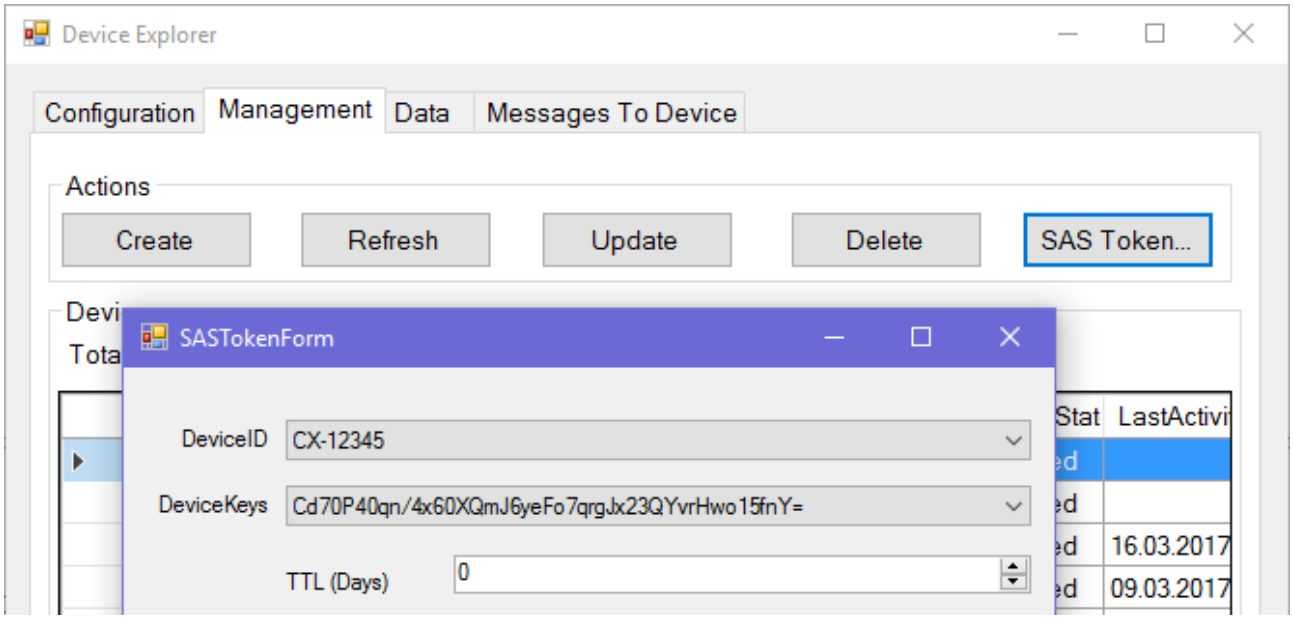


Publish

Beim Publishen von Daten an den IoT Hub muss das Topic in folgender Form angegeben werden:

devices / deviceId / messages / events / readpipe

Die DeviceId entspricht hierbei der DeviceId des registrierten Geräts, wie z. B. im Azure IoT Explorer angegeben.



Subscribe

Beim Subscribe auf Daten vom IoT Hub muss das Topic in folgender Form angegeben werden:

devices / deviceId / messages / devicebound / #

Die DeviceId entspricht hierbei der DeviceId des registrierten Geräts, wie z. B. im Azure IoT Explorer angegeben.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.0	IPC oder CX (x86, x64, ARM)	Tc3_IotBase, Tc2_Utilities (>= v3.3.19.0)

6.7 IotMqttSampleBoschIoT

Beispiel zur MQTT Kommunikation mit dem Bosch IoT Hub

In diesem Beispiel wird die Kommunikation zum Bosch IoT Hub gezeigt, welche Bestandteil der Bosch IoT Suite ist. Dieser Message Broker setzt die Verwendung von TLS und Username/Password-Authentifizierung voraus.

Ersteinrichtung Bosch IoT Hub

Informationen zur Ersteinrichtung der Bosch IoT Hub finden Sie in der offiziellen Bosch IoT Suite Dokumentation.

Topic-Struktur

Die Topic-Struktur beim Senden und Empfangen von Nachrichten ist fest vom Message Broker der Bosch IoT Suite vorgegeben.

In diesem Beispiel werden sowohl Nachrichten an den Bosch IoT Hub versendet als auch von dort empfangen. Da das Beispiel im Wesentlichen auf dem Beispiel IotMqttSampleUsingQueue [▶ 319] basiert, werden in diesem Abschnitt nur die relevanten Teile zur Verbindungsherstellung mit dem Bosch IoT Hub erklärt.

Parameter für den Verbindungsaufbau

Das folgende Code Snippet zeigt die notwendigen Parameter für einen Verbindungsaufbau mit dem Bosch IoT Hub. Im Wesentlichen handelt es sich hierbei um statische Parameter. Diese können auch im Deklarationsteil bei der Instanziierung des MQTT Client angegeben werden.

```
IF bSetParameter THEN
  bSetParameter := FALSE;
  fbMqttClient.stTLS.sCA := 'C:\TwinCAT\3.1\Config\Certificates\BoschIotHub.crt';
  fbMqttClient.sHostName:= 'mqtt.bosch-iot-hub.com';
  fbMqttClient.nHostPort:= 8883;
  fbMqttClient.sClientId:= 'CX-12345';
  fbMqttClient.sUserName:= 'com.someName_CX@t42c3e689c5c64c34b13084b9504ed3c8_hub';
  fbMqttClient.sUserPassword:= 'somePassword';
END_IF
```

Die zur Authentifizierung benötigten Parameter können auf der Bosch IoT Plattform generiert werden.

6.8 IotMqttSampleIbmWatsonIoT

Beispiel zur MQTT Kommunikation mit IBM Watson IoT

In diesem Beispiel wird die Kommunikation mit IBM Watson IoT gezeigt.

Topic-Struktur

Die Topic-Struktur beim Senden und Empfangen von Nachrichten ist fest vom IBM Watson IoT Message Broker vorgegeben.

Es werden sowohl Nachrichten an IBM Watson IoT versendet als auch von dort empfangen. Da dieses Beispiel somit im Wesentlichen auf dem Beispiel IotMqttSampleUsingQueue [▶ 319] basiert, werden in diesem Abschnitt nur die relevanten Teile zur Verbindungsherstellung erklärt.

Parameter für Verbindungsaufbau

Das folgende Code Snippet zeigt die notwendigen Parameter für einen Verbindungsaufbau mit IBM Watson IoT. Im Wesentlichen handelt es sich hierbei um statische Parameter. Diese können auch im Deklarationsteil bei der Instanziierung des MQTT Client angegeben werden.

```
IF bSetParameter THEN
  bSetParameter := FALSE;
  fbMqttClient.sHostName := 'orgid.messaging.internetofthings.ibmcloud.com';
  fbMqttClient.nHostPort := 1883;
  fbMqttClient.sClientId := 'd:orgid:IPC:deviceId';
  fbMqttClient.sUserName := 'use-token-auth';
  fbMqttClient.sUserPassword := '12342y?c12Gfq_8r12';
END_IF
```

Die Topics für den Publish und das Subscribe sind fest von IBM Watson IoT vorgegeben. Der Platzhalter „orgId“ muss mit der Organisations-ID des IBM Watson Accounts ersetzt werden. Der Platzhalter „deviceId“ wird durch die Device-ID ersetzt, so wie sie auf IBM Watson angelegt wurde.

Publish

Beim Publishen von Daten IBM Watson IoT muss das Topic in folgender Form angegeben werden:

iot-2 / evt / eventId / fmt / json

Die EventId entspricht hierbei der Event-ID, so wie sie in IBM Watson IoT konfiguriert wurde und erwartet wird. Im Falle der Nutzung eines IBM Watson Dashboards wird die Event-ID dynamisch erzeugt und kann auf der IBM Watson IoT mit einem Chart verknüpft werden.

Subscribe

Beim Subscribe auf Kommandos von IBM Watson IoT muss das Topic in folgender Form angegeben werden:

iot-2 / cmd / cmdId / fmt / json

Die CmdId entspricht hierbei die Command-ID des über IBM Watson IoT abgeschickten Kommandos an das Gerät.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.0	IPC oder CX (x86, x64, ARM)	Tc3_IotBase, Tc2_Utilities (>= v3.3.19.0)

6.9 IotMqttSampleMathworksThingspeak

In diesem Beispiel wird die Kommunikation mit der MathWorks ThingSpeak Cloud dargestellt. Da dieses Beispiel im Wesentlichen auf dem Beispiel IotMqttSampleUsingQueue [▶ 319] basiert, werden in diesem Abschnitt nur die relevanten Teile zur Verbindungsherstellung mit der ThingSpeak Cloud erklärt.

Beispielcode können Sie als Archiv hier herunterladen: https://infosys.beckhoff.com/content/1031/tf6701_tc3_iot_communication_mqtt/Resources/11990435339.zip.

MQTT Device Konfiguration

Um MQTT-Daten an ThingSpeak zu übertragen, müssen Sie Ihr MQTT Device zunächst registrieren. Melden Sie sich bei ThingSpeak mit Ihrem MathWorks Account an und wählen Sie im oberen Menü **Devices > MQTT**. Wählen Sie **Add new Device**. Benennen Sie Ihr Gerät und autorisieren Sie *publish* und *subscribe* auf dem entsprechenden ThingSpeak-Kanälen. Speichern Sie Ihre Anmeldedaten.

Parameter für Verbindungsaufbau

Das folgende Code Snippet zeigt die notwendigen Parameter für einen Verbindungsaufbau mit der MathWorks ThingSpeak Cloud. Im Wesentlichen handelt es sich hierbei um statische Parameter. Diese können auch im Deklarationsteil bei der Instanziierung des MQTT Client angegeben werden. Die im Folgenden zu nutzenden Daten <Client ID>, <Username> und <Password> werden Ihnen bei der Konfiguration eines MQTT Device auf ThingSpeak mitgeteilt.

```
IF bSetParameter THEN
  bSetParameter := FALSE;
  fbMqttClient.sHostName:= 'mqtt3.thingspeak.com';
  fbMqttClient.nHostPort:= 1883;
  fbMqttClient.sClientId:= '<Client ID as provided by MQTT Device on ThingSpeak>';
  fbMqttClient.sUsername:= '<username as provided by MQTT Device on ThingSpeak>';
  fbMqttClient.sUserPassword:= '<password as provided by MQTT Device on ThingSpeak>';
END_IF
```

ThingSpeak unterstützt ebenfalls die Absicherung der Kommunikationsverbindung über TLS. Hierbei muss das CA-Zertifikat von ThingSpeak heruntergeladen und über die TLS-Struktur beim Funktionsbaustein referenziert werden. Für weitere Informationen zum ThingSpeak CA-Zertifikat empfehlen wir die Mathworks ThingSpeak Dokumentation.

```
IF bSetParameter THEN
  bSetParameter := FALSE;
  fbMqttClient.sHostName:= 'mqtt3.thingspeak.com';
  fbMqttClient.nHostPort:= 8883;
  fbMqttClient.sClientId:= '<Client ID as provided by MQTT Device on ThingSpeak>';
  fbMqttClient.sUsername:= '<username as provided by MQTT Device on ThingSpeak>';
  fbMqttClient.sUserPassword:= '<password as provided by MQTT Device on ThingSpeak>';

  stMqttTls.sVersion:= 'tlsv1.2';
  stMqttTls.sCA:= 'C:\TwinCAT\3.1\Config\Certificates\<thingSpeakCert.pem>;
  fbMqttClient.stTLS := stMqttTls;
END_IF
```

Publish

Beim Publishen von Daten an die MathWorks ThingSpeak Cloud muss das Topic in folgender Form angegeben werden:

channels / <channelID> / publish

<channelID> entspricht hierbei der ID des Kanals, der im MathWorks ThingSpeak Portal für den Datenempfang vorgesehen und konfiguriert wurde.

Ihr Gerät muss für *publish* auf dem angegebenen Kanal autorisiert sein. Sie können die Autorisierung über **Devices > MQTT** im Top Menü ändern, wenn Sie auf der ThingSpeak-Website angemeldet sind.

Subscribe

Beim Subscriben auf Daten muss das Topic in folgender Form angegeben werden:

channels / <channelID> / subscribe / fields / <fieldKey>

<channelID> entspricht hierbei der ID des Kanals, der im MathWorks ThingSpeak Portal für den Datenempfang vorgesehen und konfiguriert wurde.

<fieldKey> entspricht dem Namen des Felds, von dem ein Wert empfangen werden soll.

Ihr Gerät muss für ein *subscribe* auf dem angegebenen Kanal autorisiert sein. Sie können die Autorisierung über **Geräte > MQTT** im oberen Menü ändern, wenn Sie auf der ThingSpeak-Website angemeldet sind

Datenformat

Die MathWorks ThingSpeak Cloud nutzt als Payload ein eigenes, string-basiertes Datenformat. Dieses Datenformat wird im oben referenzierten Beispielcode in der Funktion F_Mqtt_ThingSpeak_CreatePayloadStr() generiert und kann direkt genutzt werden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.0	IPC oder CX (x86, x64, ARM)	Tc3_IotBase, Tc2_Utilities (>= v3.3.19.0)

6.10 IotMqttSampleAzureIotHubDeviceTwin

Beispiel zur MQTT Kommunikation mit dem Device Twin des Azure IoT Hub

In diesem Beispiel wird die Kommunikation zum Device Twin gezeigt. Der Device Twin ist über den Azure IoT Hub erreichbar, welcher Bestandteil der Microsoft Azure Cloud ist. Der Message Broker ist über MQTT erreichbar und benötigt zur Authentifizierung ein sogenanntes SAS-Token, welches sich über die Azure IoT Hub Plattform generieren lässt, z.B. mithilfe des sogenannten Azure IoT Explorers.

Das folgende Beispiel stellt ebenso wie das IotMqttSampleAzureIotHub [▶ 327]-Beispiel eine Verbindung zum Azure IoT Hub her. Alle relevanten Informationen zur Verbindungsherstellung und weiteren Konfigurationsmöglichkeiten sind aus diesem Artikel zu entnehmen.

Die offizielle MQTT Dokumentation von Microsoft kann hier abgerufen werden:

https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-mqtt-support

6.11 IotMqttv5Sample

Beispiel zur MQTTv5-Kommunikation mithilfe einer Message-Queue

In diesem Beispiel wird die Kommunikation zu einem MQTT Broker unter Verwendung von MQTTv5 dargestellt. Als Voraussetzung muss der verwendete Message Broker MQTTv5 unterstützen. Es werden Nachrichten verschickt („Publish“) und empfangen („Subscribe“). Dies erfolgt in zwei Schritten. Zuerst wird anhand des Topics entschieden, welche Nachrichten empfangen werden sollen. Anschließend werden die empfangenen Nachrichten in einer Message Queue eingesammelt, um von hier aus gelesen und ausgewertet werden zu können. Das Sample verschickt und empfängt zu Demonstrationszwecken Nachrichten auf demselben Topic.

> **Handhabung der MessageQueue**
>
> Im Gegensatz zum FB_IotMqttClient Baustein für MQTTv3, müssen Sie im FB_IotMqtt5Client Baustein die Message Queue nicht mehr separat als Objekt instanziieren und mit dem Baustein verknüpfen. Anstelle dessen können Sie direkt auf den entsprechenden Ausgang des Bausteins zugreifen um mit der Message Queue zu arbeiten.

Projektstruktur

1. Erstellen Sie ein TwinCAT-Projekt mit einer SPS und fügen Sie die Tc3_IotBase und Tc3_JsonXml als Bibliotheksreferenzen an.
2. Legen Sie einen Programmbaustein an und deklarieren Sie eine Instanz von FB_IotMqtt5ClientBase [▶ 81] sowie zwei Hilfsvariablen, um den Programmablauf bei Bedarf steuern zu können.

```
PROGRAM PrgMqttCom
VAR
    fbMqttClient    : FB_IotMqtt5Client;
    bSetParameter   : BOOL := TRUE;
    bConnect        : BOOL := TRUE;
END_VAR
```

3. Deklarieren Sie für die zu verschickende MQTT-Nachricht zwei Variablen für Topic und Payload. Im Beispiel soll jede Sekunde eine Nachricht verschickt werden. Für die Nachricht werden drei imaginäre Sensorwerte verwendet, welche über die noch zu erstellende Funktion F_CreateMessage in ein JSON Dokument verpackt werden sollen.

```
    (* published message *)
    sTopicPub : STRING(255) := 'MyTopic';
    sPayloadPub : STRING(255);
    fbTimer : TON := (PT:=T#1S);
```

```
    rSensor1 : REAL;
    nSensor2 : DINT;
    bSensor3 : BOOL;
```

4. Deklarieren Sie für den Nachrichtenempfang eine Variable, die das Topic enthält, auf das sich der Client subscriben soll und zwei weitere Variablen, die Topic und Payload der zuletzt empfangenen Nachricht anzeigen.
   Die empfangenen Nachrichten sollen in einer Warteschlange (Queue) gesammelt werden, um eine nach der anderen ausgewertet werden zu können. Hierfür deklarieren Sie eine Instanz von FB_IotMqtt5MessageQueue sowie eine Instanz von FB_IotMqtt5Message.

```
    (* received message *)
    bSubscribed    : BOOL;
    sTopicSub      : STRING(255) := 'MyTopic';
    {attribute 'TcEncoding':='UTF-8'}
    sTopicRcv      : STRING(255);
    {attribute 'TcEncoding':='UTF-8'}
    sPayloadRcv    : STRING(255);
    fbMessage      : FB_IotMqtt5Message;
```

5. Im Programmteil muss der MQTT Client zyklisch getriggert werden, um den Verbindungsaufbau zum Broker, den Verbindungserhalt und den Nachrichtenempfang zu gewährleisten. Setzen Sie die Parameter der gewünschten Verbindung und initialisieren Sie den Verbindungsaufbau mit dem Übergabeparameter `bConnect := TRUE`.
   Im Beispiel werden die Parameter einmalig vor dem Client-Aufruf im Programmcode zugewiesen. Weil dies meist nur einmalig nötig ist, können die Parameter auch bereits im Deklarationsteil bei der Instanziierung des MQTT Client angegeben werden. Nicht alle Parameter müssen zugewiesen werden. Im Beispiel ist der Broker lokal. Sie können auch die IP-Adresse oder den Namen angeben.

```
IF bSetParameter THEN
    bSetParameter := FALSE;
    fbMqttClient.sHostName := 'localhost';
    fbMqttClient.nHostPort := 1883;
END_IF

fbMqttClient.Execute(bConnect);
```

6. Sobald die Verbindung zum Broker aufgebaut wird, soll sich der Client auf ein bestimmtes Topic anmelden. Ebenso soll jede Sekunde eine Nachricht versandt werden.
   Im Beispiel ist `sTopicPub = sTopicSub`, sodass ein Loopback entsteht. In anderen Applikationen unterscheiden sich die Topics typischerweise.

```
IF fbMqttClient.bConnected THEN
    IF NOT bSubscribed THEN
        bSubscribed := fbMqttClient.Subscribe(sTopic:=sTopicSub, eQoS:=TcIotMqttQos.AtMostOnceDe
livery);
    END_IF
    fbTimer(IN:=TRUE);
    IF fbTimer.Q THEN // publish new payload every second
        fbTimer(IN:=FALSE);
        rSensor1 := rSensor1 + 0.1;
        nSensor2 := nSensor2 + 1;
        bSensor3 := NOT bSensor3;
        sPayloadPub := F_CreateMessage(rSensor1, nSensor2, bSensor3);
        fbMqttClient.Publish(sTopic:= sTopicPub,
pPayload:= ADR(sPayloadPub), nPayloadSize:= LEN2(ADR(sPayloadPub))
+1, eQoS:= TcIotMqttQos.AtMostOnceDelivery, bRetain:= FALSE, bQueue:= FALSE );
    END_IF
END_IF
```

7. Der zyklische Aufruf vom MQTT Client sorgt für den Empfang der Nachrichten. Der Client empfängt alle Nachrichten, auf deren Topic er sich zuvor beim Broker angemeldet hat und legt diese in der Message Queue ab. Sobald Nachrichten verfügbar sind, rufen Sie die Methode Dequeue() auf und erhalten über das Nachrichtenobjekt fbMessage Zugriff auf die Nachrichteneigenschaften wie Topic und Payload.

```
IF fbMqttClient.fbMessageQueue.nQueuedMessages > 0 THEN
    IF fbMqttClient.fbMessageQueue.Dequeue(fbMessage:=fbMessage) THEN
        fbMessage.GetTopic(pTopic:=ADR(sTopicRcv), nTopicSize:=SIZEOF(sTopicRcv));
        fbMessage.GetPayload(pPayload:=ADR(sPayloadRcv), nPayloadSize:=SIZEOF(sPayloadRcv), bSet
NullTermination:=FALSE);
    END_IF
END_IF
```

8. Bei obiger Implementierung der Nachrichtenauswertung wird pro Zyklus eine empfangene Nachricht ausgewertet. Falls mehrere Nachrichten in der Message-Queue gesammelt wurden, verteilt sich deren Auswertung entsprechend auf mehrere Zyklen.

9. Zum Schluss fügen Sie eine neue Funktion namens F_CreateMessage hinzu. Die Funktion soll den Rückgabewert STRING(255) besitzen und ist mit der folgenden Funktions-Signatur zu versehen:

```
FUNCTION F_CreateMessage : STRING(255)
VAR_INPUT
  Sensor1 : REAL;
  Sensor2 : DINT;
  Sensor3 : BOOL;
END_VAR
VAR
  dtTimestamp : DATE_AND_TIME;
  timeAsFileTime : T_FILETIME64;
  fbJson : FB_JsonSaxWriter;
END_VAR
```

10. Die Funktion erzeugt zunächst einen Zeitstempel basierend auf der Funktion F_GetSystemTime und verpackt anschließend die übergebenen Sensorwerte und den Zeitstempel in ein JSON Dokument.

```
timeAsFileTime := F_GetSystemTime();
dtTimestamp := FILETIME64_TO_DT( timeAsFileTime );
fbJson.StartObject();
fbJson.AddKey('Timestamp');
fbJson.AddDateTime(dtTimestamp);
fbJson.AddKey('Values');
fbJson.StartObject();
fbJson.AddKey('Sensor1');
fbJson.AddReal(Sensor1);
fbJson.AddKey('Sensor2');
fbJson.AddDint(Sensor2);
fbJson.AddKey('Sensor3');
fbJson.AddBool(Sensor3);
fbJson.EndObject();
fbJson.EndObject();
F_CreateMessage := fbJson.GetDocument();
fbJson.ResetDocument();
```

Weitere Schritte

Das Beispiel kann abgewandelt werden für Applikationen, in denen sich auf mehrere Topics angemeldet wird. Dann werden MQTT-Nachrichten mit verschiedenen Topics empfangen. Die Auswertung der Nachrichten könnten Sie wie folgt für diesen Anwendungsfall erweitern:

```
VAR
    (* received payload for each subscribed topic *)
    sPayloadRcv1 : STRING(255);
    sPayloadRcv2 : STRING(255);
END_VAR
VAR CONSTANT
    (* subscriptions *)
    sTopicSub1 : STRING(255) := 'my first topic';
    sTopicSub2 : STRING(255) := 'my second topic';
END_VAR
-------------------------------------------------------------------

IF fbMqttClient.fbMessageQueue.nQueuedMessages > 0 THEN
    IF fbMqttClient.fbMessageQueue.Dequeue(fbMessage:=fbMessage) THEN
        IF fbMessage.CompareTopic(sTopic:=sTopicSub1) THEN
            fbMessage.GetPayload(pPayload:=ADR(sPayloadRcv1), nPayloadSize:=SIZEOF(sPayloadRcv1), bS
etNullTermination:=FALSE);
        ELSIF fbMessage.CompareTopic(sTopic:=sTopicSub2) THEN
            fbMessage.GetPayload(pPayload:=ADR(sPayloadRcv2), nPayloadSize:=SIZEOF(sPayloadRcv2), bS
etNullTermination:=FALSE);
        END_IF
    END_IF
END_IF
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4026.0	IPC oder CX (x86, x64, ARM)	Tc3_IotBase (>= v3.4.2.0), Tc2_Utilities (>= v3.4.4.0)

6.12 IotMqttv5LastWillSample

Beispiel zur Verwendung von MQTTv5 UserProperties

In diesem Beispiel wird die Kommunikation zu einem MQTT Broker unter Verwendung von MQTTv5 dargestelt. Hierbei wird insbesondere der LastWill Mechanismus von MQTTv5 demonstriert. Als Voraussetzung muss der verwendete Message Broker MQTTv5 unterstützen. Der grundlegende Ablauf des Samples stellt sich wie folgt dar:

- Es gibt ein SPS-Projekt, welches auf Basis von MQTTv5 eine Verbindung zu einem lokalen Message Broker herstellt und Nachrichten an diesen published. Gleichzeitig subscribed sich die Applikation auf dasselbe Topic um die gesendeten Nachrichten wieder zu empfangen. Der Aufbau des Programms entspricht hierbei dem Sample IotMqttv5Sample [▶ 333].
- Beim Verbindungsaufbau wird eine LastWill Nachricht spezifiziert, sowie einige Properties für die LastWill Nachricht gesetzt. Der LastWill wird dann beim Connect mit dem Message Broker an diesen übertragen und im Falle eines Disconnects an interessierte Clients übermittelt.

Das folgende Code Snippet zeigt noch einmal den relevanten Teil zur Spezifizierung der LastWill Nachricht, sowie der damit einhergehenden Properties. Diese Codestelle wird bei der (nur einmalig erfolgenden) Initialisierung der Verbindungsparameter verwendet.

```
fbMqttClient.stWill.sTopic := 'MyLastWillTopic';
fbMqttClient.stWill.sContentType := 'MyContentType';
fbMqttClient.stWill.eQoS := TcIotMqttQoS.ExactlyOnceDelivery;
fbMqttClient.stWill.fbPayload.SetData(ADR(sLastWillMsg), SIZEOF(sLastWillMsg));
fbMqttClient.stWill.fbUserProperties.AddUserProperty('MyFirst', 'UserProperty');
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4026.0	IPC oder CX (x86, x64, ARM)	Tc3_IotBase (>= v3.4.2.0), Tc2_Utilities (>= v3.4.4.0)

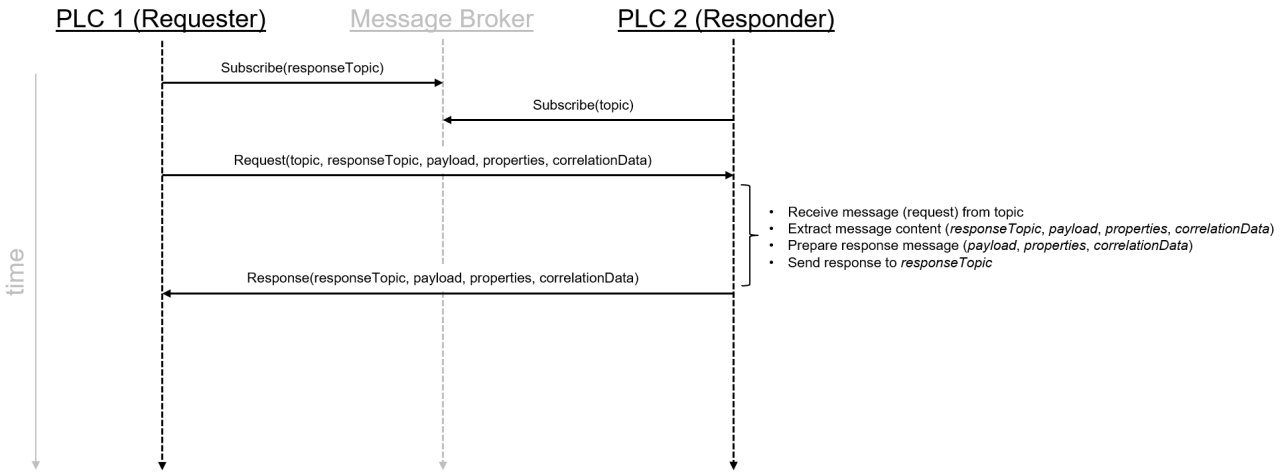
6.13 IotMqttv5ReqResSample

Beispiel zur Verwendung von MQTTv5 Request/Response

In diesem Beispiel wird die Kommunikation zu einem MQTT Broker unter Verwendung von MQTTv5 dargestellt. Hierbei wird insbesondere der Request/Response [▶ 34] Mechanismus von MQTTv5 demonstriert. Als Voraussetzung muss der verwendete Message Broker MQTTv5 unterstützen. Der grundlegende Ablauf des Samples stellt sich wie folgt dar:

- Es gibt ein SPS-Projekt, welches die Request-Funktion demonstriert und auch die Response von der Gegenstelle empfängt und auswertet. Das Projekt subscribed sich zunächst auf das sogenannte Response-Topic (auf dem es eine Antwort von der Gegenstelle erwartet) und verschickt anschliessend den Request, welcher den Namen des Reponse-Topics beinhaltet. Hierbei werden auch erweiterte Eigenschaften der MQTTv5 Nachricht, wie z.B. UserProperties oder CorrelationData, demonstriert.
- Ein zweites SPS-Projekt ist für den Empfang des Requests und den Versand einer entsprechenden Response an das Response-Topic zuständig. Hierbei subscribed sich der MQTT-Client zunächst auf das Topic auf dem der Request eingeht und bereitet bei Empfang eines Requests die entsprechende Response vor. Das Response-Topic, sowie die erweiterten Eigenschaften der MQTTv5 Nachricht werden hierbei aus der empfangenen Nachricht extrahiert und für die Response verwendet.

Das folgende Schaubild veranschaulicht noch einmal den grundlegenden Ablauf der Kommunikation. Weitere Informationen zu einzelnen Codezeilen finden Sie direkt in den Kommentaren des entsprechenden Sample-Downloads.



Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4026.0	IPC oder CX (x86, x64, ARM)	Tc3_IotBase (>= v3.4.2.0), Tc2_Utilities (>= v3.4.4.0)

6.14 IotMqttv5UserPropsSample

Beispiel zur Verwendung von MQTTv5 UserProperties

In diesem Beispiel wird die Kommunikation zu einem MQTT Broker unter Verwendung von MQTTv5 dargestellt. Hierbei wird insbesondere der UserProperties Mechanismus von MQTTv5 demonstriert. Als Voraussetzung muss der verwendete Message Broker MQTTv5 unterstützen. Der grundlegende Ablauf des Samples stellt sich wie folgt dar:

- Es gibt ein SPS-Projekt, welches auf Basis von MQTTv5 eine Verbindung zu einem lokalen Message Broker herstellt und Nachrichten an diesen published. Gleichzeitig subscribed sich die Applikation auf dasselbe Topic um die gesendeten Nachrichten wieder zu empfangen.
- Der Aufbau des Programms entspricht dem Sample IotMqttv5Sample [▶ 333]. Die Nachrichten enthalten neben dem eigentlichen Payload auch UserProperties, deren Handhabung sowohl beim Nachrichtenversand als auch -empfang demonstriert wird.

Das folgende Code Snippet zeigt noch einmal den relevanten Teil zur Spezifizierung der User Properties an einer zu versendenden Nachricht.

```
fbPubProps.sContentType := sContentType;
fbPubProps.nMsgExpiryInterval := 7;
fbPubProps.bPayloadUtf8 := TRUE;
fbPubProps.ClearUserProperties();
FOR m:=1 TO 10 DO
  hrPropSet := fbPubProps.AddUserProperty(aUserName[m], aUserValue[m]);
  IF FAILED(hrPropSet) THEN
    EXIT;
  END_IF
END_FOR
```

In diesem Beispiel werden somit 10 User Properties, deren Key/Value-Werte aus den Array aUserName und aUserValue entnommen werden, zu einer zu versendenden Nachricht hinzugefügt. Die Übergabe der User Properties an die zu versendende Nachricht erfolgt dann als Eingabeparameter an der Publish() Methode.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4026.0	IPC oder CX (x86, x64, ARM)	Tc3_IotBase (>= v3.4.2.0), Tc2_Utilities (>= v3.4.4.0)

6.15 JsonXmlSamples

6.15.1 Tc3JsonXmlSampleXmlDomWriter

Dieses Beispiel veranschaulicht, wie ein XML-Dokument programmatisch auf Basis von DOM erstellt werden kann. Als Basis wird der Funktionsbaustein FB_XmlDomParser verwendet.

Deklarationsbereich

```
PROGRAM MAIN
VAR
  fbXml : FB_XmlDomParser;
  objRoot : SXmlNode;
  objMachines : SXmlNode;
  objMachine : SXmlNode;
  objControllers : SXmlNode;
  objController : SXmlNode;
  objAttribute : SXmlAttribute;
  sXmlString : STRING(1000);
  bCreate : BOOL := FALSE;
  bSave : BOOL := TRUE;
  nLength : UDINT;
  newAttr : SXmlAttribute;
END_VAR
```

Implementierungsbereich

Der Implementierungsbereich zeigt verschiedene Möglichkeiten auf, wie ein XML-Dokument erstellt werden kann.

```
IF bCreate THEN
  (* Create an empty XML document *)
  objRoot := fbXml.GetDocumentNode();

  (* Create a new XML node 'Machines' and add to the empty document *)
  objMachines := fbXml.AppendNode(objRoot, 'Machines');

  (* Create a new XML node 'Machine' and add an attribute to this node. Append node to 'Machines' *)
  objMachine := fbXml.AppendNode(objMachines, 'Machine');
  objAttribute := fbXml.AppendAttribute(objMachine, 'Name', 'Wilde Nelli');

  (* Create a new XML node 'Controllers' and add to the 'Machine' node *)
  objControllers := fbXml.AppendNode(objMachine, 'Controllers');

  (* Create a new XML node 'Controller' and add some attributes. Append node to 'Controllers'. *)
  objController := fbXml.AppendChild(objControllers, 'Controller', 'CX5120', FALSE);
  objAttribute := fbXml.AppendAttribute(objController, 'Type', 'EPC');
  objAttribute := fbXml.AppendAttribute(objController, 'OS', 'Windows Embedded Compact 7');

  (* Create a new XML node 'Controller' and add some attributes. Append node to 'Controllers'. *)
  objController := fbXml.AppendChild(objControllers, 'Controller', 'CX2040', FALSE);
  objAttribute := fbXml.AppendAttribute(objController, 'Type', 'EPC');
  objAttribute := fbXml.AppendAttribute(objController, 'OS', 'Windows Embedded Standard 7');

  (* Create a new XML node 'Controller' and add some attributes. Append node to 'Controllers'. *)
  objController := fbXml.AppendChild(objControllers, 'Controller', 'C6015', FALSE);
  objAttribute := fbXml.AppendAttribute(objController, 'Type', 'IPC');
  objAttribute := fbXml.AppendAttribute(objController, 'OS', 'Windows 10 IoT Enterprise');

  (* Create a new XML node 'Machine' and add an attribute to this node. Append node to 'Machines' *)
  objMachine := fbXml.AppendNode(objMachines, 'Machine');
  objAttribute := fbXml.AppendAttribute(objMachine, 'Name', 'Stanze Oscar');

  (* Create a new XML node 'Controllers' and add to the 'Machine' node *)
  objControllers := fbXml.AppendNode(objMachine, 'Controllers');

  (* Create a new XML node 'Controller' and add some attributes. Append node to 'Controllers'. *)
  objController := fbXml.AppendChild(objControllers, 'Controller', 'C6017', FALSE);
  objAttribute := fbXml.AppendAttribute(objController, 'Type', 'IPC');
  objAttribute := fbXml.AppendAttribute(objController, 'OS', 'Windows 10 IoT Enterprise');
  newAttr := fbXml.InsertAttribute(objController, objAttribute, 'AddAttribute');
  fbXml.SetAttribute(newAttr, 'Hola');

  (* Retrieve XML document and store in a variable of data type STRING(1000) *)
```

```
  nLength := fbXml.CopyDocument(sXmlString, SIZEOF(sXmlString));
  bCreate := FALSE;
END_IF
```

6.15.2 Tc3JsonXmlSampleXmlDomReader

Dieses Beispiel veranschaulicht, wie ein XML-Dokument programmatisch auf Basis von DOM durchlaufen werden kann. Als Basis wird der Funktionsbaustein FB_XmlDomParser verwendet.

Deklarationsbereich

```
PROGRAM MAIN
VAR
  fbXml : FB_XmlDomParser;
  xmlDoc : SXmlNode;
  xmlMachines : SXmlNode;
  xmlMachine1 : SXmlNode;
  xmlMachine2 : SXmlNode;
  xmlIterator : SXmlIterator;
  xmlMachineNode : SXmlNode;
  xmlMachineNodeValue : STRING;
  xmlMachineAttributeRef : SXmlAttribute;
  xmlMachine1Attribute : SXmlAttribute;
  xmlMachine2Attribute : SXmlAttribute;
  sMachine1Name : STRING;
  sMachine2Name : STRING;
  nMachineAttribute : DINT;
  nMachine1Attribute : DINT;
  nMachine2Attribute : DINT;
  sMessageToParse : STRING(255) := '<Machines><Machine Type="1" Test="3">Wilde Nelli</
Machine><Machine Type="2">Huber8</Machine></Machines>';
END_VAR
```

Implementierungsbereich

Der Implementierungsbereich zeigt verschiedene Möglichkeiten auf, wie man ein XML-Dokument parsen kann.

```
(* Load XML content *)
xmlDoc := fbXml.ParseDocument(sMessageToParse);

(* Parse XML nodes - Option 1 *)
xmlMachines := fbXml.ChildByName(xmlDoc, 'Machines');
xmlMachine1 := fbXml.ChildByAttribute(xmlMachines, 'Type', '1');
xmlMachine2 := fbXml.ChildByAttributeAndName(xmlMachines, 'Type', '2', 'Machine');

(* Parse XML nodes - Option 2 *)
xmlMachines := fbXml.ChildByName(xmlDoc, 'Machines');
xmlMachineNode := fbXml.Children(xmlMachines, xmlIterator);
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlMachineNode := fbXml.Node(xmlIterator);
  xmlMachineNodeValue := fbXml.NodeText(xmlMachineNode);
  xmlMachineNode := fbXml.Children(xmlMachines, xmlIterator);
END_WHILE

(* Parse XML nodes - Option 3 *)
xmlMachines := fbXml.ChildByName(xmlDoc, 'Machines');
xmlIterator := fbXml.Begin(xmlMachines);
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlMachineNode := fbXml.Node(xmlIterator);
  xmlMachineNodeValue := fbXml.NodeText(xmlMachineNode);
  xmlIterator := fbXml.Next(xmlIterator);
  xmlIterator := fbXml.End(xmlMachines);
END_WHILE

(* Parse XML attributes - Option 1*)
xmlMachine1Attribute := fbXml.Attribute(xmlMachine1, 'Type');
xmlMachine2Attribute := fbXml.Attribute(xmlMachine2, 'Type');

(* Parse XML attributes - Option 2*)
xmlIterator := fbXml.AttributeBegin(xmlMachine1);
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlMachineAttributeRef := fbXml.AttributeFromIterator(xmlIterator);
  nMachineAttribute := fbXml.AttributeAsInt(xmlMachineAttributeRef);
  xmlIterator := fbXml.Next(xmlIterator);
END_WHILE
```

```
(* Retrieve node values *)
sMachine1Name := fbXml.NodeText(xmlMachine1);
sMachine2Name := fbXml.NodeText(xmlMachine2);

(* Retrieve attribute values *)
nMachine1Attribute := fbXml.AttributeAsInt(xmlMachine1Attribute);
nMachine2Attribute := fbXml.AttributeAsInt(xmlMachine2Attribute);
```

6.15.3 Tc3JsonXmlSampleJsonDomReader

Dieses Beispiel veranschaulicht, wie eine JSON-Nachricht programmatisch auf Basis von DOM durchlaufen werden kann. Als Basis wird der Funktionsbaustein FB_JsonDomParser verwendet.

Deklarationsbereich

```
PROGRAM MAIN
VAR
  fbJson     : FB_JsonDomParser;
  jsonDoc    : SJsonValue;
  jsonProp   : SJsonValue;
  jsonValue  : SJsonValue;
  bHasMember : BOOL;
  sMessage   : STRING(255) := '{"serialNumber":"G030PT028191AC4R","batteryVoltage":"1547mV","clickTy
pe":"SINGLE"}';
  stReceivedData : ST_ReceivedData;
END_VAR
```

Implementierungsbereich

Durch die Methode ParseDocument() wird die JSON-Nachricht in den DOM-Tree geladen. Anschließend kann mit der Methode HasMember() überprüft werden, ob ein bestimmtes Property enthalten ist. Über die Methode FindMember() wird das Property selektiert und über GetString() dessen Value extrahiert.

```
jsonDoc := fbJson.ParseDocument(sMessage);
bHasMember := fbJson.HasMember(jsonDoc, 'serialNumber');
IF (bHasMember) THEN
  bHasMember := FALSE;
  jsonProp := fbJson.FindMember(jsonDoc, 'serialNumber');
  stReceivedData.serialNumber := fbJson.GetString(jsonProp);
END_IF

bHasMember := fbJson.HasMember(jsonDoc, 'batteryVoltage');
IF (bHasMember) THEN
  bHasMember := FALSE;
  jsonProp := fbJson.FindMember(jsonDoc, 'batteryVoltage');
  stReceivedData.batteryVoltage := fbJson.GetString(jsonProp);
END_IF

bHasMember := fbJson.HasMember(jsonDoc, 'clickType');
IF (bHasMember) THEN
  bHasMember := FALSE;
  jsonProp := fbJson.FindMember(jsonDoc, 'clickType');
  stReceivedData.clickType := fbJson.GetString(jsonProp);
END_IF
```

Die Verwendung der Methode HasMember() ist nicht zwingend erforderlich, da die Methode FindMember() bereits 0 zurückliefert, wenn ein Property nicht gefunden wurde. Der oben dargestellte Code kann also auch wie folgt implementiert werden:

```
jsonDoc := fbJson.ParseDocument(sMessage);

jsonProp := fbJson.FindMember(jsonDoc, 'serialNumber');
IF (jsonProp <> 0) THEN
  stReceivedData.serialNumber := fbJson.GetString(jsonProp);
END_IF

jsonProp := fbJson.FindMember(jsonDoc, 'batteryVoltage');
IF (jsonProp <> 0) THEN
  stReceivedData.batteryVoltage := fbJson.GetString(jsonProp);
END_IF

jsonProp := fbJson.FindMember(jsonDoc, 'clickType');
IF (jsonProp <> 0) THEN
  stReceivedData.clickType := fbJson.GetString(jsonProp);
END_IF
```

Verschachtelte JSON-Objekte

Bei verschachtelten JSON-Objekten ist die Herangehensweise ähnlich. Dadurch, dass sich das gesamte Dokument im DOM befindet, kann darin einfach navigiert werden. Gegeben sei ein JSON-Objekt, das sich wie folgt darstellt:

```
sMessage : STRING(255) := '{"Values":{"serial":"G030PT028191AC4R"}}';
```

Das gesuchte Property befindet sich in dem Unterobjekt „Values“. Der folgende Code zeigt, wie das Property extrahiert werden kann.

```
jsonDoc := fbJson.ParseDocument(sMessage);
bHasMember := fbJson.HasMember(jsonDoc, 'Values');
IF (bHasMember) THEN
  bHasMember := FALSE;
  jsonProp := fbJson.FindMember(jsonDoc, 'Values');
  IF jsonProp <> 0 THEN
    jsonSerial := fbJson.FindMember(jsonProp, 'serial');
    stReceivedData.serialNumber := fbJson.GetString(jsonSerial);
  END_IF
END_IF
```

6.15.4 Tc3JsonXmlSampleJsonSaxWriter

Beispiel zum Erstellen von JSON-Dokumenten via SAX Writer

Dieses Beispiel veranschaulicht, wie eine JSON-Nachricht über den SAX-Mechanismus erstellt werden kann. Als Basis wird der Funktionsbaustein FB_JsonSaxWriter verwendet.

Deklarationsbereich

```
PROGRAM MAIN
VAR
  dtTimestamp : DATE_AND_TIME := DT#2017-04-04-12:42:42;
  fbJson      : FB_JsonSaxWriter;
  sJsonDoc    : STRING(255);
END_VAR
```

Implementierungsbereich

Der SAX-Mechanismus durchläuft ein zu erstellendes JSON-Dokument sequentiell, d. h. die entsprechenden Elemente werden der Reihe nach durchlaufen und erstellt.

```
fbJson.StartObject();
fbJson.AddKey('Timestamp');
fbJson.AddDateTime(dtTimestamp);
fbJson.AddKey('Values');
fbJson.StartObject();
fbJson.AddKey('Sensor1');
fbJson.AddReal(42.42);
fbJson.AddKey('Sensor2');
fbJson.AddDint(42);
fbJson.AddKey('Sensor3');
fbJson.AddBool(TRUE);
fbJson.EndObject();
fbJson.EndObject();
sJsonDoc := fbJson.GetDocument();
fbJson.ResetDocument();
```

Resultierende JSON-Nachricht

```
{
  "Timestamp": "2017-04-04T12:42:42",
  "Values": {
    "Sensor1": 42.42,
    "Sensor2": 42,
    "Sensor3": true
  }
}
```

6.15.5 Tc3JsonXmlSampleJsonSaxReader

Beispiel zum Parsen von JSON-Dokumenten via SAX Reader

Dieses Beispiel veranschaulicht, wie eine JSON-Nachricht programmatisch durchlaufen werden kann. Als Basis wird der Funktionsbaustein FB_JsonSaxReader verwendet.

Deklarationsbereich

```
PROGRAM MAIN
VAR
  fbJson     : FB_JsonSaxReader;
  pJsonParse : JsonSaxHandler;
  sJsonDoc   : STRING(255) := '{"Values":
{"Timestamp":"2017-04-04T12:42:42","Sensor1":42.42,"Sensor2":42}}';
END_VAR
```

Implementierungsbereich

Durch den Aufruf der Methode Parse(), die Übergabe der JSON-Nachricht als STRING und den Interface-Pointer auf eine Funktionsbaustein-Instanz, die das Interface ITcJsonSaxHandler implementiert, werden der SAX Reader aktiviert und die entsprechenden Callback-Methoden durchlaufen.

```
fbJson.Parse(sJson := sJsonDoc, ipHdl := pJsonParse);
```

Callback-Methoden

Die Callback-Methoden werden an der Instanz des Funktionsbausteins, der das Interface ITcJsonSaxHandler implementiert, aufgerufen. Jede Callback-Methode repräsentiert ein „gefundenes“ Element in der JSON-Nachricht. Zum Beispiel wird die Callback-Methode OnStartObject() aufgerufen, sobald eine geöffnete geschweifte Klammer detektiert wurde. Laut der oben genannten Beispiel-JSON-Nachricht werden also die folgenden Callback-Methoden in dieser Reihenfolge durchlaufen:

1. OnStartObject(), aufgrund der ersten geöffneten geschweiften Klammer
2. OnKey(), aufgrund des Properties "Values"
3. OnStartObject(), aufgrund der zweiten geöffneten geschweiften Klammer
4. OnKey(), aufgrund des Properties "Timestamp"
5. OnString(), aufgrund des Werts von Property "Timestamp"
6. OnKey(), aufgrund des Properties "Sensor1"
7. OnLreal(), aufgrund des Werts von Property "Sensor1“"
8. OnKey(), aufgrund des Properties "Sensor2"
9. OnUdint(), aufgrund des Werts von Property "Sensor2"
10. OnEndObject(), aufgrund der ersten geschlossenen geschweiften Klammer
11. OnEndObject(), aufgrund der zweiten geschlossenen geschweiften Klammer

Innerhalb der Callback-Methoden wird der aktuelle Zustand über eine Instanz des Enums E_JsonStates definiert und gespeichert. Hierüber kann auch ermittelt werden, ob es sich um eine gültige JSON-Nachricht handelt. Wenn zum Beispiel die Callback-Methode OnLreal() aufgerufen wird und sich der Zustand nicht im erwarteten State 70 (JSON_STATE_ONLREAL) befindet, kann an die Methode der Rückgabewert S_FALSE zurückgegeben werden. Der SAX Reader beendet dann automatisch die weitere Verarbeitung.

6.15.6 Tc3JsonXmlSampleJsonDataType

Beispiel zum automatischen Konvertieren von Strukturen in eine JSON-Nachricht

Dieses Beispiel veranschaulicht, wie eine Datenstruktur in eine JSON-Nachricht (und umgekehrt) konvertiert werden kann. Bei der Konvertierung wird der Aufbau einer Struktur eins-zu-eins in ein entsprechendes JSON-Äquivalent überführt. Über SPS-Attribute an den Member-Variablen der Struktur können zusätzlich Metadaten angelegt werden.

Aufbau der zu konvertierenden Datenstruktur

```
TYPE ST_Values :
STRUCT

  {attribute 'Unit' := 'm/s'}
  {attribute 'DisplayName' := 'Speed'}
  Sensor1 : REAL;
```

```
  {attribute 'Unit' := 'V'}
  {attribute 'DisplayName' := 'Voltage'}
  Sensor2 : DINT;

  {attribute 'Unit' := 'A'}
  {attribute 'DisplayName' := 'Current'}
  Sensor3 : DINT;

END_STRUCT
END_TYPE
```

Deklarationsbereich

```
PROGRAM MAIN
VAR
  dtTimestamp    : DATE_AND_TIME := DT#2017-04-04-12:42:42;
  fbJson         : FB_JsonSaxWriter;
  fbJsonDataType : FB_JsonReadWriteDataType;
  sJsonDoc       : STRING(255);
  sJsonDoc2      : STRING(2000);
  stValues       : ST_Values;
END_VAR
```

Implementierungsbereich

Ausgehend von der Instanz fbJson des Funktionsbausteins FB_JsonSaxWriter werden zwei Wege zum Generieren der JSON-Nachricht gezeigt. Bei einer JSON-Nachricht mit nicht mehr als 255 Zeichen kann die Methode GetDocument() verwendet werden. Bei größeren JSON-Nachrichten muss hingegen die Methode CopyDocument() verwendet werden.

```
fbJson.ResetDocument();
fbJson.StartObject();
fbjson.AddKeyDateTime('Timestamp', dtTimestamp);
fbJsonDataType.AddJsonKeyValueFromSymbol(fbJson, 'Values', 'ST_Values', SIZEOF(stValues), ADR(stValu
es));
fbJsonDataType.AddJsonKeyPropertiesFromSymbol(fbJson, 'MetaData', 'ST_Values', 'Unit|DisplayName');
fbJson.EndObject();
sJsonDoc := fbJson.GetDocument();
fbJson.CopyDocument(sJsonDoc2, SIZEOF(sJsonDoc2));
```

Resultierende JSON-Nachricht

```
{
  "Timestamp": "2017-04-04T12:42:42",
  "Values": {
    "Sensor1": 0.0,
    "Sensor2": 0,
    "Sensor3": 0
  },
  "MetaData": {
    "Sensor1": {
      "Unit": "m/s",
      "DisplayName": "Speed"
    },
    "Sensor2": {
      "Unit": "V",
      "DisplayName": "Voltage"
    },
    "Sensor3": {
      "Unit": "A",
      "DisplayName": "Current"
    }
  }
}
```

Alternative

Als Alternative kann auch die Methode AddJsonValueFromSymbol() verwendet werden, um aus einer Datenstruktur direkt ein JSON-Format zu erzeugen.

```
fbJson.ResetDocument();
fbJsonDataType.AddJsonValueFromSymbol(fbJson, 'ST_Values', SIZEOF(stValues), ADR(stValues));
sJsonDoc := fbJson.GetDocument();
fbJson.CopyDocument(sJsonDoc2, SIZEOF(sJsonDoc2));
```

Das resultierende JSON-Objekt sieht wie folgt aus:

```
{
  "Sensor1": 0.0,
  "Sensor2": 0,
  "Sensor3": 0
}
```

Konvertierung einer JSON-Nachricht zurück in eine Datenstruktur

Die obigen Beispiele zeigen, wie auf einfache Art und Weise aus einer Datenstruktur ein JSON-Objekt erzeugt werden kann. Für den umgekehrten Weg, also das Extrahieren von Werten aus einem (empfangenen) JSON-Objekt zurück in eine Datenstruktur, gibt es ebenfalls eine entsprechende Methode in der Tc3_JsonXml-Bibliothek. Der Aufruf der Methode SetSymbolFromJson() ermöglicht diesen Anwendungsfall.

```
fbJsonDataType.SetSymbolFromJson(someJson, 'ST_Values', SIZEOF(stValuesReceive),
ADR(stValuesReceive));
```

Die String-Variable sJsonDoc2 enthält das JSON-Objekt, das durch den Aufruf der Methode in die Strukturinstanz stValuesReceive überführt wird.

Zieldatenstruktur

Die Zieldatenstruktur muss zum Aufbau des JSON-Dokuments passen. Ansonsten liefert SetSymbolFromJson() FALSE zurück.

7 Anhang

7.1 Beispielkonfigurationen

7.1.1 Mosquitto

Im Folgenden finden Sie einige Beispielkonfigurationen für den Mosquitto Message Broker. Die Konfigurationen wurden auf Basis der offiziellen Mosquitto Dokumentation erstellt. Die Konfigurationsdatei (mosquitto.conf) des Brokers finden Sie im Installationsverzeichnis des Brokers (zum Beispiel C:\Program Files\mosquitto). Diese kann mit einem Texteditor Ihrer Wahl geöffnet und bearbeitet werden, zum Beispiel Notepad.

Wir empfehlen vor Änderungen an der Konfigurationsdatei ein Backup zu erstellen. Alternativ können Sie auch zusätzliche Konfigurationsdateien erstellen und diese beim Starten des Brokers über die Kommandozeile laden:

```
mosquitto.exe -c mosquitto.conf
```

Zu Debugging- und Test-Zwecken empfehlen wir den Start des Brokers in einer Kommandozeile, da hier auch die Verbose Ausgabe aktiviert werden kann:

```
mosquitto.exe -c mosquitto.conf -v
```

Mosquitto Start in der Kommandozeile

Bitte stellen Sie vor dem Starten des Mosquitto Brokers aus der Kommandozeile sicher, dass kein aktiver Mosquitto Prozess auf Ihrem System läuft. Dies kann zum Beispiel der Fall sein, wenn der Mosquitto Broker als Windows-Service installiert und gestartet wurde. Bitte überprüfen Sie daher den Windows Task Manager und die Windows Diensteverwaltung.

Im Folgenden finden Sie nun einige Beispielkonfigurationen für unterschiedliche Anwendungsfälle. Diese Konfigurationen wurden auf Basis des Mosquitto Message Brokers in der Version 2.0.15 erstellt und sind ebenfalls in unserem TF6701_Samples Repository auf GitHub verfügbar.

Ungesicherte Verbindung

Der folgende Inhalt der Konfigurationsdatei konfiguriert den Broker für eine ungesicherte Verbindung ohne TLS und ohne Benutzerauthentifizierung.

```
listener 1883
allow_anonymous true
```

Ungesicherte Verbindung mit Benutzerauthentifizierung

Der folgende Inhalt der Konfigurationsdatei konfiguriert den Broker für eine ungesicherte Verbindung ohne TLS, jedoch mit Benutzerauthentifizierung.

```
listener 1883
allow_anonymous false
password_file C:\Program Files\mosquitto\users.pwd
```

Die mit dem Parameter password_file spezifizierte Benutzerdatenbank kann mit dem Tool mosquitto_passwd erstellt werden, welches sich ebenfalls im Installationsverzeichnis des Brokers befindet. Der folgende Aufruf erzeugt eine neue Benutzerdatenbank und fügt den Benutzer MyUser1 mit dem Password SecurePassword hinzu:

```
mosquitto_passwd.exe -b -c "C:\Program Files\mosquitto\users.pwd" MyUser1 SecurePassword
```

Durch die Verwendung des Parameters -b kann ein Password direkt in dem Aufruf übergeben werden. Der Parameter -c erzeugt eine neue Password-Datei. Soll ein neuer Benutzer zu einer existierenden Datei hinzugefügt werden, so entfällt dieser Parameter.

```
mosquitto_passwd.exe -b "C:\Program Files\mosquitto\users.pwd" MyUser2 AnotherSecurePassword
```

Klartext-Übermittlung des Passwords

Bitte beachten Sie, dass im Falle einer unsicheren, d.h. nicht-verschlüsselten MQTT-Verbindung, das Password bei der Benutzerauthentifizierung im Klartext übermittelt wird. Wir empfehlen daher die Verschlüsselung der MQTT-Kommunikation mittels TLS, siehe Beispielkonfigurationen unten.

Ungesicherte Verbindung mit Benutzerauthentifizierung und ACL

Der folgende Inhalt der Konfigurationsdatei konfiguriert den Broker für eine ungesicherte Verbindung ohne TLS, jedoch mit Benutzerauthentifizierung und unter Verwendung einer Access Control List (ACL), welche den Benutzerzugriff auf bestimmte Topics reguliert.

```
listener 1883
allow_anonymous false
password_file C:\Program Files\mosquitto\users.pwd
acl_file C:\Program Files\mosquitto\userAccess.acl
```

Wie bereits im vorherigen Beispiel beschrieben, kann die als password_file referenzierte Benutzerdatenbank mit Hilfe des Tools mosquitto_passwd erzeugt werden. Die als acl_file referenzierte Zugriffslistendatei ist eine reine Textdatei, welche nach einem bestimmten Format aufgebaut wird. Dieses Format ist ebenfalls in der Mosquitto Dokumentation näher beschrieben. Die folgende Datei zeigt exemplarisch, wie eine Zugriffsliste für zwei Benutzer aussehen könnte. Jedem Benutzer wird hierbei Zugriff auf seinen eigenen Topic-Bereich gewährt. Ein Administrator-Benutzer soll jedoch Zugriff auf alle Topics bekommen.

```
user Admin
topic #
user MyUser1
topic users/MyUser1/#
user MyUser2
topic users/MyUser2/#
```

Zugriff auf ein nicht erlaubtes Topic

Bitte beachten Sie, dass der Client bei Zugriff (Publish/Subscribe) auf ein nicht erlaubtes Topic keinerlei Rückmeldung vom Mosquitto Message Broker bekommt. Aus Client-Sicht scheint der Vorgang erfolgreich zu sein, es werden vom Broker jedoch außer den regulären MQTT Kommando-Paketen keinerlei Nachrichten an den Client übermittelt bzw. weitergeleitet.

TLS-Verbindung mit PSK

Der folgende Inhalt der Konfigurationsdatei konfiguriert den Broker für eine gesicherte Verbindung mit TLS unter Verwendung eines Pre-Shared Keys (PSK). Es wird keine Benutzerauthentifizierung konfiguriert.

```
listener 8883
allow_anonymous true
psk_hint TwinCATrocks
psk_file C:\Program Files\mosquitto\myKeys.psk
```

Die als psk_file referenzierte Datei enthält dann eine Liste mit Pre-Shared Keys. Es handelt sich hierbei um eine reine Textdatei, in welcher die PSKs im Format identity:key zeilenweise aufgelistet sind. Der Key wird hierbei im Hexadezimalformat angegeben. Zum Beispiel:

```
MyIdentity1:ab123456789cd
MyIdentity2:ef987654321ab
```

Der Parameter psk_hint ist wichtig, da er für den Listener das TLS-PSK aktiviert.

TLS-Verbindung mit Zertifikat

Der folgende Inhalt der Konfigurationsdatei konfiguriert den Broker für eine gesicherte Verbindung mit TLS unter Verwendung eines Zertifikats. Es wird keine Benutzerauthentifizierung konfiguriert.

Dieses Beispiel geht davon aus, dass Sie eine Certificate Authority (CA) besitzen, welche Serverzertifikate ausstellen kann. Es gibt diverse Tutorials im Web, welche zum Beispiel demonstrieren, wie Sie mittels OpenSSL eine solche CA erzeugen und entsprechende Zertifikate ausstellen können.

```
listener 8883
allow_anonymous true
cafile C:\ca\certs\fullChain.pem
crlfile C:\ca\crl\intermediateCA.crl
certfile C:\Program Files\mosquitto\certs\mosquitto.pem
keyfile C:\Program Files\mosquitto\certs\mosquitto.key
```

TLS-Verbindung mit Zertifikat und Benutzerauthentifizierung

Der folgende Inhalt der Konfigurationsdatei konfiguriert den Broker für eine gesicherte Verbindung mit TLS unter Verwendung eines Zertifikats. Zusätzlich wird eine Benutzerauthentifizierung unter Angabe der Benutzerdatenbankdatei konfiguriert.

Dieses Beispiel geht davon aus, dass Sie eine Certificate Authority (CA) besitzen, welche Serverzertifikate ausstellen kann. Es gibt diverse Tutorials im Web, welche zum Beispiel demonstrieren, wie Sie mittels OpenSSL eine solche CA erzeugen und entsprechende Zertifikate ausstellen können.

```
listener 8883
allow_anonymous false
password_file C:\Program Files\mosquitto\users.pwd
cafile C:\ca\certs\fullChain.pem
crlfile C:\ca\crl\intermediateCA.crl
certfile C:\Program Files\mosquitto\certs\mosquitto.pem
keyfile C:\Program Files\mosquitto\certs\mosquitto.key
```

Clients die versuchen eine Verbindung zu diesem Broker aufzubauen müssen ein gültiges Zertifikat besitzen, welches von derselben Certificate Authority ausgestellt wurde.

TLS-Verbindung mit Zertifikat, Benutzerauthentifizierung und ACL

Der folgende Inhalt der Konfigurationsdatei konfiguriert den Broker für eine gesicherte Verbindung mit TLS unter Verwendung eines Zertifikats. Zusätzlich wird eine Benutzerauthentifizierung unter Angabe der Benutzerdatenbankdatei konfiguriert, sowie eine Access Control List (ACL), welche verschiedene Zugriffsrechte für die Benutzer konfiguriert (siehe Beispiel oben).

Dieses Beispiel geht davon aus, dass Sie eine Certificate Authority (CA) besitzen, welche Serverzertifikate ausstellen kann. Es gibt diverse Tutorials im Web, welche zum Beispiel demonstrieren, wie Sie mittels OpenSSL eine solche CA erzeugen und entsprechende Zertifikate ausstellen können.

```
listener 8883
allow_anonymous false
use_identity_as_username true
password_file C:\Program Files\mosquitto\users.pwd
acl_file C:\Program Files\mosquitto\userAccess.acl
cafile C:\ca\certs\fullChain.pem
crlfile C:\ca\crl\intermediateCA.crl
certfile C:\Program Files\mosquitto\certs\mosquitto.pem
keyfile C:\Program Files\mosquitto\certs\mosquitto.key
```

Clients die versuchen eine Verbindung zu diesem Broker aufzubauen müssen ein gültiges Zertifikat besitzen, welches von derselben Certificate Authority ausgestellt wurde.

Aufbauend auf diesem Beispiel können Sie nun auch den Parameter use_identity_as_username verwenden, welcher bewirkt, dass der CommonName aus dem Clientzertifikat als Benutzername verwendet wird.

7.2 Fehlercodes

Der Funktionsbaustein FB_IotMqttClient [▶ 57] setzt im Fehlerfall den Ausgang bError und zeigt mit hrErrorCode den Fehler an. Im Abschnitt „ADS Return Codes [▶ 348]“ werden alle Fehler aufgelistet.

Zusätzlich zeigt der Ausgang eConnectionState jederzeit den Zustand der Verbindung vom Client zum MQTT Broker an. Die Enumeration bietet hierbei folgende mögliche Zustände:

```
TYPE ETcIotMqttClientState :
(
   MQTT_ERR_CONN_PENDING:=-1,
   MQTT_ERR_SUCCESS:=0,
   MQTT_ERR_NOMEM:=1,
   MQTT_ERR_PROTOCOL:=2,
   MQTT_ERR_INVAL:=3,
   MQTT_ERR_NO_CONN:=4,
   MQTT_ERR_CONN_REFUSED:=5,
   MQTT_ERR_NOT_FOUND:=6,
   MQTT_ERR_CONN_LOST:=7,
   MQTT_ERR_TLS:=8,
   MQTT_ERR_PAYLOAD_SIZE:=9,
   MQTT_ERR_NOT_SUPPORTED:=10,
   MQTT_ERR_AUTH:=11,
   MQTT_ERR_ACL_DENIED:=12,
```

```
   MQTT_ERR_UNKNOWN:=13,
   MQTT_ERR_ERRNO:=14,
   MQTT_ERR_EAI:=15,
   MQTT_ERR_PROXY:=16
) DINT;
END_TYPE
```

7.3 ADS Return Codes

Gruppierung der Fehlercodes:
Globale Fehlercodes: 0x0000 [▶ 348]... (0x9811_0000 ...)
Router Fehlercodes: 0x0500 [▶ 348]... (0x9811_0500 ...)
Allgemeine ADS Fehler: 0x0700 [▶ 349]... (0x9811_0700 ...)
RTime Fehlercodes: 0x1000 [▶ 351]... (0x9811_1000 ...)

Globale Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x0	0	0x98110000	ERR_NOERROR	Kein Fehler.
0x1	1	0x98110001	ERR_INTERNAL	Interner Fehler.
0x2	2	0x98110002	ERR_NORTIME	Keine Echtzeit.
0x3	3	0x98110003	ERR_ALLOCLOCKEDMEM	Zuweisung gesperrt - Speicherfehler.
0x4	4	0x98110004	ERR_INSERTMAILBOX	Postfach voll – Es konnte die ADS Nachricht nicht versendet werden. Reduzieren der Anzahl der ADS Nachrichten pro Zyklus bringt Abhilfe.
0x5	5	0x98110005	ERR_WRONGRECEIVEHMSG	Falsches HMSG.
0x6	6	0x98110006	ERR_TARGETPORTNOTFOUND	Ziel-Port nicht gefunden – ADS Server ist nicht gestartet oder erreichbar.
0x7	7	0x98110007	ERR_TARGETMACHINENOTFOUND	Zielrechner nicht gefunden – AMS Route wurde nicht gefunden.
0x8	8	0x98110008	ERR_UNKNOWNCMDID	Unbekannte Befehl-ID.
0x9	9	0x98110009	ERR_BADTASKID	Ungültige Task-ID.
0xA	10	0x9811000A	ERR_NOIO	Kein IO.
0xB	11	0x9811000B	ERR_UNKNOWNAMSCMD	Unbekannter AMS-Befehl.
0xC	12	0x9811000C	ERR_WIN32ERROR	Win32 Fehler.
0xD	13	0x9811000D	ERR_PORTNOTCONNECTED	Port nicht verbunden.
0xE	14	0x9811000E	ERR_INVALIDAMSLENGTH	Ungültige AMS-Länge.
0xF	15	0x9811000F	ERR_INVALIDAMSNETID	Ungültige AMS Net ID.
0x10	16	0x98110010	ERR_LOWINSTLEVEL	Installations-Level ist zu niedrig –TwinCAT 2 Lizenzfehler.
0x11	17	0x98110011	ERR_NODEBUGINTAVAILABLE	Kein Debugging verfügbar.
0x12	18	0x98110012	ERR_PORTDISABLED	Port deaktiviert – TwinCAT System Service nicht gestartet.
0x13	19	0x98110013	ERR_PORTALREADYCONNECTED	Port bereits verbunden.
0x14	20	0x98110014	ERR_AMSSYNC_W32ERROR	AMS Sync Win32 Fehler.
0x15	21	0x98110015	ERR_AMSSYNC_TIMEOUT	AMS Sync Timeout.
0x16	22	0x98110016	ERR_AMSSYNC_AMSERROR	AMS Sync Fehler.
0x17	23	0x98110017	ERR_AMSSYNC_NOINDEXINMAP	Keine Index-Map für AMS Sync vorhanden.
0x18	24	0x98110018	ERR_INVALIDAMSPORT	Ungültiger AMS-Port.
0x19	25	0x98110019	ERR_NOMEMORY	Kein Speicher.
0x1A	26	0x9811001A	ERR_TCPSEND	TCP Sendefehler.
0x1B	27	0x9811001B	ERR_HOSTUNREACHABLE	Host nicht erreichbar.
0x1C	28	0x9811001C	ERR_INVALIDAMSFRAGMENT	Ungültiges AMS Fragment.
0x1D	29	0x9811001D	ERR_TLSSEND	TLS Sendefehler – Secure ADS Verbindung fehlgeschlagen.
0x1E	30	0x9811001E	ERR_ACCESSDENIED	Zugriff Verweigert – Secure ADS Zugriff verweigert.

Router Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x500	1280	0x98110500	ROUTERERR_NOLOCKEDMEMORY	Lockierter Speicher kann nicht zugewiesen werden.
0x501	1281	0x98110501	ROUTERERR_RESIZEMEMORY	Die Größe des Routerspeichers konnte nicht geändert werden.
0x502	1282	0x98110502	ROUTERERR_MAILBOXFULL	Das Postfach hat die maximale Anzahl der möglichen Meldungen erreicht.
0x503	1283	0x98110503	ROUTERERR_DEBUGBOXFULL	Das Debug Postfach hat die maximale Anzahl der möglichen Meldungen erreicht.
0x504	1284	0x98110504	ROUTERERR_UNKNOWNPORTTYPE	Der Porttyp ist unbekannt.
0x505	1285	0x98110505	ROUTERERR_NOTINITIALIZED	Router ist nicht initialisiert.
0x506	1286	0x98110506	ROUTERERR_PORTALREADYINUSE	Die Portnummer ist bereits vergeben.
0x507	1287	0x98110507	ROUTERERR_NOTREGISTERED	Der Port ist nicht registriert.
0x508	1288	0x98110508	ROUTERERR_NOMOREQUEUES	Die maximale Portanzahl ist erreicht.
0x509	1289	0x98110509	ROUTERERR_INVALIDPORT	Der Port ist ungültig.
0x50A	1290	0x9811050A	ROUTERERR_NOTACTIVATED	Der Router ist nicht aktiv.
0x50B	1291	0x9811050B	ROUTERERR_FRAGMENTBOXFULL	Das Postfach hat die maximale Anzahl für fragmentierte Nachrichten erreicht.
0x50C	1292	0x9811050C	ROUTERERR_FRAGMENTTIMEOUT	Fragment Timeout aufgetreten.
0x50D	1293	0x9811050D	ROUTERERR_TOBEREMOVED	Port wird entfernt.

Allgemeine ADS Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x700	1792	0x98110700	ADSERR_DEVICE_ERROR	Allgemeiner Gerätefehler.
0x701	1793	0x98110701	ADSERR_DEVICE_SRVNOTSUPP	Service wird vom Server nicht unterstützt.
0x702	1794	0x98110702	ADSERR_DEVICE_INVALIDGRP	Ungültige Index-Gruppe.
0x703	1795	0x98110703	ADSERR_DEVICE_INVALIDOFFSET	Ungültiger Index-Offset.
0x704	1796	0x98110704	ADSERR_DEVICE_INVALIDACCESS	Lesen oder Schreiben nicht gestattet.
0x705	1797	0x98110705	ADSERR_DEVICE_INVALIDSIZE	Parametergröße nicht korrekt.
0x706	1798	0x98110706	ADSERR_DEVICE_INVALIDDATA	Ungültige Daten-Werte.
0x707	1799	0x98110707	ADSERR_DEVICE_NOTREADY	Gerät nicht betriebsbereit.
0x708	1800	0x98110708	ADSERR_DEVICE_BUSY	Gerät beschäftigt.
0x709	1801	0x98110709	ADSERR_DEVICE_INVALIDCONTEXT	Ungültiger Kontext vom Betriebssystem - Kann durch Verwendung von ADS Bausteinen in unterschiedlichen Tasks auftreten. Abhilfe kann die Multitasking-Syncronisation in der SPS geben.
0x70A	1802	0x9811070A	ADSERR_DEVICE_NOMEMORY	Nicht genügend Speicher.
0x70B	1803	0x9811070B	ADSERR_DEVICE_INVALIDPARM	Ungültige Parameter-Werte.
0x70C	1804	0x9811070C	ADSERR_DEVICE_NOTFOUND	Nicht gefunden (Dateien,...).
0x70D	1805	0x9811070D	ADSERR_DEVICE_SYNTAX	Syntax-Fehler in Datei oder Befehl.
0x70E	1806	0x9811070E	ADSERR_DEVICE_INCOMPATIBLE	Objekte stimmen nicht überein.
0x70F	1807	0x9811070F	ADSERR_DEVICE_EXISTS	Objekt ist bereits vorhanden.
0x710	1808	0x98110710	ADSERR_DEVICE_SYMBOLNOTFOUND	Symbol nicht gefunden.
0x711	1809	0x98110711	ADSERR_DEVICE_SYMBOLVERSIONINVALID	Symbol-Version ungültig – Kann durch einen Online-Change auftreten. Erzeuge einen neuen Handle.
0x712	1810	0x98110712	ADSERR_DEVICE_INVALIDSTATE	Gerät (Server) ist im ungültigen Zustand.
0x713	1811	0x98110713	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransMode nicht unterstützt.
0x714	1812	0x98110714	ADSERR_DEVICE_NOTIFYHNDINVALID	Notification Handle ist ungültig.
0x715	1813	0x98110715	ADSERR_DEVICE_CLIENTUNKNOWN	Notification-Client nicht registriert.
0x716	1814	0x98110716	ADSERR_DEVICE_NOMOREHDLS	Keine weiteren Handles verfügbar.
0x717	1815	0x98110717	ADSERR_DEVICE_INVALIDWATCHSIZE	Größe der Notification zu groß.
0x718	1816	0x98110718	ADSERR_DEVICE_NOTINIT	Gerät nicht initialisiert.
0x719	1817	0x98110719	ADSERR_DEVICE_TIMEOUT	Gerät hat einen Timeout.
0x71A	1818	0x9811071A	ADSERR_DEVICE_NOINTERFACE	Interface Abfrage fehlgeschlagen.
0x71B	1819	0x9811071B	ADSERR_DEVICE_INVALIDINTERFACE	Falsches Interface angefordert.
0x71C	1820	0x9811071C	ADSERR_DEVICE_INVALIDCLSID	Class-ID ist ungültig.
0x71D	1821	0x9811071D	ADSERR_DEVICE_INVALIDOBJID	Object-ID ist ungültig.
0x71E	1822	0x9811071E	ADSERR_DEVICE_PENDING	Anforderung steht aus.
0x71F	1823	0x9811071F	ADSERR_DEVICE_ABORTED	Anforderung wird abgebrochen.
0x720	1824	0x98110720	ADSERR_DEVICE_WARNING	Signal-Warnung.
0x721	1825	0x98110721	ADSERR_DEVICE_INVALIDARRAYIDX	Ungültiger Array-Index.
0x722	1826	0x98110722	ADSERR_DEVICE_SYMBOLNOTACTIVE	Symbol nicht aktiv.
0x723	1827	0x98110723	ADSERR_DEVICE_ACCESSDENIED	Zugriff verweigert.
0x724	1828	0x98110724	ADSERR_DEVICE_LICENSENOTFOUND	Fehlende Lizenz.
0x725	1829	0x98110725	ADSERR_DEVICE_LICENSEEXPIRED	Lizenz abgelaufen.
0x726	1830	0x98110726	ADSERR_DEVICE_LICENSEEXCEEDED	Lizenz überschritten.
0x727	1831	0x98110727	ADSERR_DEVICE_LICENSEINVALID	Lizenz ungültig.
0x728	1832	0x98110728	ADSERR_DEVICE_LICENSESYSTEMID	Lizenzproblem: System-ID ist ungültig.
0x729	1833	0x98110729	ADSERR_DEVICE_LICENSENOTIMELIMIT	Lizenz nicht zeitlich begrenzt.
0x72A	1834	0x9811072A	ADSERR_DEVICE_LICENSEFUTUREISSUE	Lizenzproblem: Zeitpunkt in der Zukunft.
0x72B	1835	0x9811072B	ADSERR_DEVICE_LICENSETIMETOLONG	Lizenz-Zeitraum zu lang.
0x72C	1836	0x9811072C	ADSERR_DEVICE_EXCEPTION	Exception beim Systemstart.
0x72D	1837	0x9811072D	ADSERR_DEVICE_LICENSEDUPLICATED	Lizenz-Datei zweimal gelesen.
0x72E	1838	0x9811072E	ADSERR_DEVICE_SIGNATUREINVALID	Ungültige Signatur.
0x72F	1839	0x9811072F	ADSERR_DEVICE_CERTIFICATEINVALID	Zertifikat ungültig.
0x730	1840	0x98110730	ADSERR_DEVICE_LICENSEOEMNOTFOUND	Public Key vom OEM nicht bekannt.
0x731	1841	0x98110731	ADSERR_DEVICE_LICENSERESTRICTED	Lizenz nicht gültig für diese System.ID.
0x732	1842	0x98110732	ADSERR_DEVICE_LICENSEDEMODENIED	Demo-Lizenz untersagt.
0x733	1843	0x98110733	ADSERR_DEVICE_INVALIDFNCID	Funktions-ID ungültig.
0x734	1844	0x98110734	ADSERR_DEVICE_OUTOFRANGE	Außerhalb des gültigen Bereiches.
0x735	1845	0x98110735	ADSERR_DEVICE_INVALIDALIGNMENT	Ungültiges Alignment.

Hex	Dec	HRESULT	Name	Beschreibung
0x736	1846	0x98110736	ADSERR_DEVICE_LICENSEPLATFORM	Ungültiger Plattform Level.
0x737	1847	0x98110737	ADSERR_DEVICE_FORWARD_PL	Kontext – Weiterleitung zum Passiv-Level.
0x738	1848	0x98110738	ADSERR_DEVICE_FORWARD_DL	Kontext – Weiterleitung zum Dispatch-Level.
0x739	1849	0x98110739	ADSERR_DEVICE_FORWARD_RT	Kontext – Weiterleitung zur Echtzeit.
0x740	1856	0x98110740	ADSERR_CLIENT_ERROR	Clientfehler.
0x741	1857	0x98110741	ADSERR_CLIENT_INVALIDPARM	Dienst enthält einen ungültigen Parameter.
0x742	1858	0x98110742	ADSERR_CLIENT_LISTEMPTY	Polling-Liste ist leer.
0x743	1859	0x98110743	ADSERR_CLIENT_VARUSED	Var-Verbindung bereits im Einsatz.
0x744	1860	0x98110744	ADSERR_CLIENT_DUPLINVOKEID	Die aufgerufene ID ist bereits in Benutzung.
0x745	1861	0x98110745	ADSERR_CLIENT_SYNCTIMEOUT	Timeout ist aufgetreten – Die Gegenstelle antwortet nicht im vorgegebenen ADS Timeout. Die Routeneinstellung der Gegenstelle kann falsch konfiguriert sein.
0x746	1862	0x98110746	ADSERR_CLIENT_W32ERROR	Fehler im Win32 Subsystem.
0x747	1863	0x98110747	ADSERR_CLIENT_TIMEOUTINVALID	Ungültiger Client Timeout-Wert.
0x748	1864	0x98110748	ADSERR_CLIENT_PORTNOTOPEN	Port nicht geöffnet.
0x749	1865	0x98110749	ADSERR_CLIENT_NOAMSADDR	Keine AMS Adresse.
0x750	1872	0x98110750	ADSERR_CLIENT_SYNCINTERNAL	Interner Fehler in Ads-Sync.
0x751	1873	0x98110751	ADSERR_CLIENT_ADDHASH	Überlauf der Hash-Tabelle.
0x752	1874	0x98110752	ADSERR_CLIENT_REMOVEHASH	Schlüssel in der Tabelle nicht gefunden.
0x753	1875	0x98110753	ADSERR_CLIENT_NOMORESYM	Keine Symbole im Cache.
0x754	1876	0x98110754	ADSERR_CLIENT_SYNCRESINVALID	Ungültige Antwort erhalten.
0x755	1877	0x98110755	ADSERR_CLIENT_SYNCPORTLOCKED	Sync Port ist verriegelt.
0x756	1878	0x98110756	ADSERR_CLIENT_REQUESTCANCELLED	Die Anfrage wurde abgebrochen.

RTime Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x1000	4096	0x98111000	RTERR_INTERNAL	Interner Fehler im Echtzeit-System.
0x1001	4097	0x98111001	RTERR_BADTIMERPERIODS	Timer-Wert nicht gültig.
0x1002	4098	0x98111002	RTERR_INVALIDTASKPTR	Task-Pointer hat den ungültigen Wert 0 (null).
0x1003	4099	0x98111003	RTERR_INVALIDSTACKPTR	Stack-Pointer hat den ungültigen Wert 0 (null).
0x1004	4100	0x98111004	RTERR_PRIOEXISTS	Die Request Task Priority ist bereits vergeben.
0x1005	4101	0x98111005	RTERR_NOMORETCB	Kein freier TCB (Task Control Block) verfügbar. Maximale Anzahl von TCBs beträgt 64.
0x1006	4102	0x98111006	RTERR_NOMORESEMAS	Keine freien Semaphoren zur Verfügung. Maximale Anzahl der Semaphoren beträgt 64.
0x1007	4103	0x98111007	RTERR_NOMOREQUEUES	Kein freier Platz in der Warteschlange zur Verfügung. Maximale Anzahl der Plätze in der Warteschlange beträgt 64.
0x100D	4109	0x9811100D	RTERR_EXTIRQALREADYDEF	Ein externer Synchronisations-Interrupt wird bereits angewandt.
0x100E	4110	0x9811100E	RTERR_EXTIRQNOTDEF	Kein externer Sync-Interrupt angewandt.
0x100F	4111	0x9811100F	RTERR_EXTIRQINSTALLFAILED	Anwendung des externen Synchronisierungs-Interrupts ist fehlgeschlagen.
0x1010	4112	0x98111010	RTERR_IRQLNOTLESSOREQUAL	Aufruf einer Service-Funktion im falschen Kontext
0x1017	4119	0x98111017	RTERR_VMXNOTSUPPORTED	Intel VT-x Erweiterung wird nicht unterstützt.
0x1018	4120	0x98111018	RTERR_VMXDISABLED	Intel VT-x Erweiterung ist nicht aktiviert im BIOS.
0x1019	4121	0x98111019	RTERR_VMXCONTROLSMISSING	Fehlende Funktion in Intel VT-x Erweiterung.
0x101A	4122	0x9811101A	RTERR_VMXENABLEFAILS	Aktivieren von Intel VT-x schlägt fehl.

Spezifische positive HRESULT Return Codes:

HRESULT	Name	Beschreibung
0x0000_0000	S_OK	Kein Fehler.
0x0000_0001	S_FALSE	Kein Fehler. Bsp.: erfolgreiche Abarbeitung, bei der jedoch ein negatives oder unvollständiges Ergebnis erzielt wurde.
0x0000_0203	S_PENDING	Kein Fehler. Bsp.: erfolgreiche Abarbeitung, bei der jedoch noch kein Ergebnis vorliegt.
0x0000_0256	S_WATCHDOG_TIMEOUT	Kein Fehler. Bsp.: erfolgreiche Abarbeitung, bei der jedoch eine Zeitüberschreitung eintrat.

TCP Winsock-Fehlercodes

Hex	Dec	Name	Beschreibung
0x274C	10060	WSAETIMEDOUT	Verbindungs Timeout aufgetreten - Fehler beim Herstellen der Verbindung, da die Gegenstelle nach einer bestimmten Zeitspanne nicht ordnungsgemäß reagiert hat, oder die hergestellte Verbindung konnte nicht aufrecht erhalten werden, da der verbundene Host nicht reagiert hat.
0x274D	10061	WSAECONNREFUSED	Verbindung abgelehnt - Es konnte keine Verbindung hergestellt werden, da der Zielcomputer dies explizit abgelehnt hat. Dieser Fehler resultiert normalerweise aus dem Versuch, eine Verbindung mit einem Dienst herzustellen, der auf dem fremden Host inaktiv ist—das heißt, einem Dienst, für den keine Serveranwendung ausgeführt wird.
0x2751	10065	WSAEHOSTUNREACH	Keine Route zum Host - Ein Socketvorgang bezog sich auf einen nicht verfügbaren Host.
Weitere Winsock-Fehlercodes: Win32-Fehlercodes			

7.4 Fehlerdiagnose

Das folgende Kapitel gibt nützliche Informationen zur Fehlerdiagnose, wenn Anwendungsszenarien nicht so funktionieren wie erwartet.

Verhalten	Kategorie	Beschreibung
Verbindung zum Message Broker kann nicht hergestellt werden	Verbindungsaufbau	Überprüfen Sie, ob der Message Broker grundsätzlich von dem System aus erreichbar ist, auf dem die Tc3_IotBase ausgeführt wird (z.B. mit einem anderen MQTT-Client). Falls der Message Broker nicht erreichbar ist, überprüfen Sie Ihre Firewall-Einstellungen. Für eine MQTT-Kommunikation muss auf dem MQTT-Client (dem Rechner der die Tc3_IotBase ausführt) standardmäßig der ausgehende TCP-Port 1883 bzw. 8883 (bei Verwendung von TLS) geöffnet sein. Auf der Seite des Message Brokers müssen diese Ports entsprechend eingehend geöffnet sein. Die Port-Konfiguration kann bei Verwendung der Tc3_IotBase über den Eingabeparameter nHostPort am Funktionsbaustein FB_IotMqttClient [▶ 57] überprüft werden.
Verbindung mit AWS IoT kann nicht hergestellt werden	Verbindungsaufbau	Überprüfen Sie, ob die AWS IoT URL grundsätzlich von dem System aus erreichbar ist, auf dem die Tc3_IotBase ausgeführt wird (z.B. mit einem anderen MQTT-Client). Dies gibt Ihnen auch die Gelegenheit die Zertifikate für die Verbindung auf Gültigkeit zu überprüfen. Falls der andere MQTT-Client keine Verbindung herstellen kann, überprüfen Sie auf der AWS IoT Management-Webseite, ob die Zertifikate gültig und aktiv sind. Falls AWS IoT nicht erreichbar ist, überprüfen Sie Ihre Firewall-Einstellungen. Für eine MQTT-Kommunikation mit AWS IoT muss auf dem MQTT-Client (dem Rechner der die Tc3_IotBase ausführt) standardmäßig der ausgehende TCP-Port 8883 geöffnet sein. Die Port-Konfiguration kann bei Verwendung der Tc3_IotBase über den Eingabeparameter nHostPort am Funktionsbaustein FB_IotMqttClient [▶ 57] überprüft werden.
Verbindung mit Azure IoT Hub kann nicht hergestellt werden	Verbindungsaufbau	Überprüfen Sie, ob die AWS IoT URL grundsätzlich von dem System aus erreichbar ist, auf dem die Tc3_IotBase ausgeführt wird (z.B. mit einem anderen MQTT-Client). Dies gibt Ihnen auch die Gelegenheit das CA-Zertifikat für die Verbindung auf Gültigkeit zu überprüfen. Falls der andere MQTT-Client keine Verbindung herstellen kann, überprüfen Sie, ob Sie ein gültiges CA-Zertifikat verwenden. Falls der Azure IoT Hub nicht erreichbar ist, überprüfen Sie Ihre Firewall-Einstellungen. Für eine MQTT-Kommunikation mit dem Azure IoT Hub muss auf dem MQTT-Client (dem Rechner der die Tc3_IotBase ausführt) standardmäßig der ausgehende TCP-Port 8883 geöffnet sein. Die Port-Konfiguration kann bei Verwendung der Tc3_IotBase über den Eingabeparameter nHostPort am Funktionsbaustein FB_IotMqttClient [▶ 57] überprüft werden.

Verhalten	Kategorie	Beschreibung
Nach einem CRL Update kann die Verbindung nicht aufgebaut werden. Die Variable eConnectionState zeigt den folgenden Wert an: MQTT_ERR_TLS_VERIFY_FAIL	Verbindungsaufbau	Das Zertifikat des Message Brokers wurde revoked. Bitte wenden Sie sich in diesem Fall an den Betreiber des Message Brokers für weitere Informationen.
Nach zwischenzeitlichem Verwenden eines ungültigen Topics (bspw. eine Wildcard an der falschen Stelle „testtopic#“) toggelt die Verbindung des MQTT-Clients zum Broker	Verbindungsaufbau	Nach Verwenden eines ungültigen Topics muss TwinCAT neugestartet werden. Bitte achten Sie darauf, dass keine ungültigen Topics verwendet werden.
Nachrichten kommen nicht im Azure IoT Hub an	Publish	Überprüfen Sie, ob Sie die Nachrichten an das richtige Topic publishen. Die Topic-Struktur ist beim Azure IoT Hub fest vorgegeben und obliegt einem festen Namensschema. Überprüfen Sie, ob Sie gültige MQTT-Einstellungen verwenden und diese vom Azure IoT Hub unterstützt werden. Konsultieren Sie unsere Hinweise zum Verbindungsaufbau mit dem Azure IoT Hub oder die MSDN-Dokumentation für weitere Informationen.
Nachrichten kommen nicht bei IBM Watson IoT an	Publish	Überprüfen Sie, ob Sie die Nachrichten an das richtige Topic publishen. Die Topic-Struktur ist IBM Watson IoT fest vorgegeben und obliegt einem festen Namensschema. Überprüfen Sie, ob Sie gültige MQTT-Einstellungen verwenden und diese von IBM Watson IoT unterstützt werden. Konsultieren Sie unsere Hinweise zum Verbindungsaufbau mit IBM Watson IoT oder die IBM Watson Dokumentation für weitere Informationen.
Nachrichten-Inhalt wird vom Subscriber nicht als gültige JSON-Nachricht erkannt	Publish	Überprüfen Sie, ob Sie ein gültiges JSON Dokument verschicken. Unsere Tc3_JsonXml-Bibliothek unterstützt Sie beim Erzeugen von gültigen JSON-Dokumenten. Überprüfen Sie, ob Sie beim Aufrufen der Publish-Methode die richtige Längeninformation (nPayloadSize) übergeben und sich nicht zu viele Daten (null) hinter dem eigentlichen Payload befinden.
Es werden keine Properties für das zugehörige Event auf IBM Watson IoT erkannt	Publish	Überprüfen Sie, ob Sie beim Aufrufen der Publish-Methode die richtige Längeninformation (nPayloadSize) übergeben und sich nicht zu viele Daten (null) hinter dem eigentlichen Payload befinden, sodass IBM Watson die Nachricht gegebenenfalls nicht als gültige JSON-Nachricht erkennen kann.
Node-RED meldet bei der Konvertierung von eingehenden MQTT Nachrichten an der JSON Funktion "Ignored non-object payload"	Publish	Überprüfen Sie, ob Sie beim Aufrufen der Publish-Methode die richtige Längeninformation (nPayloadSize) übergeben und sich nicht zu viele Daten (null) hinter dem eigentlichen Payload befinden, sodass Node-RED die Nachricht gegebenenfalls nicht als gültige JSON-Nachricht erkennen kann.

Verhalten	Kategorie	Beschreibung
Die Verbindung zum Message Broker wird zwischendurch plötzlich getrennt.	Subscribe	Überprüfen Sie, ob auf den Topics, auf die Sie sich subscribed haben, eventuell eine Nachricht empfangen wurde, welche die maximale Nachrichtengröße (cMaxSizeOfMqttMessage [▶ 117]) überschreitet. In diesem Fall trennt der MQTT Client Treiber die Verbindung zum Broker.

7.5 Support und Service

Beckhoff und seine weltweiten Partnerfirmen bieten einen umfassenden Support und Service, der eine schnelle und kompetente Unterstützung bei allen Fragen zu Beckhoff Produkten und Systemlösungen zur Verfügung stellt.

Downloadfinder

Unser Downloadfinder beinhaltet alle Dateien, die wir Ihnen zum Herunterladen anbieten. Sie finden dort Applikationsberichte, technische Dokumentationen, technische Zeichnungen, Konfigurationsdateien und vieles mehr.

Die Downloads sind in verschiedenen Formaten erhältlich.

Beckhoff Niederlassungen und Vertretungen

Wenden Sie sich bitte an Ihre Beckhoff Niederlassung oder Ihre Vertretung für den lokalen Support und Service zu Beckhoff Produkten!

Die Adressen der weltweiten Beckhoff Niederlassungen und Vertretungen entnehmen Sie bitte unserer Internetseite: www.beckhoff.com

Dort finden Sie auch weitere Dokumentationen zu Beckhoff Komponenten.

Beckhoff Support

Der Support bietet Ihnen einen umfangreichen technischen Support, der Sie nicht nur bei dem Einsatz einzelner Beckhoff Produkte, sondern auch bei weiteren umfassenden Dienstleistungen unterstützt:

- Support
- Planung, Programmierung und Inbetriebnahme komplexer Automatisierungssysteme
- umfangreiches Schulungsprogramm für Beckhoff Systemkomponenten

Hotline:	+49 5246 963-157
E-Mail:	support@beckhoff.com

Beckhoff Service

Das Beckhoff Service-Center unterstützt Sie rund um den After-Sales-Service:

- Vor-Ort-Service
- Reparaturservice
- Ersatzteilservice
- Hotline-Service

Hotline:	+49 5246 963-460
E-Mail:	service@beckhoff.com

Beckhoff Unternehmenszentrale

Beckhoff Automation GmbH & Co. KG

Hülshorstweg 20
33415 Verl
Deutschland

Telefon:	+49 5246 963-0
E-Mail:	info@beckhoff.com
Internet:	www.beckhoff.com

Mehr Informationen:
www.beckhoff.de/tf6701.html

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Deutschland
Telefon: +49 5246 9630
info@beckhoff.com
www.beckhoff.com