

Manual | EN

TF6701

TwinCAT 3 | IoT Communication (MQTT)

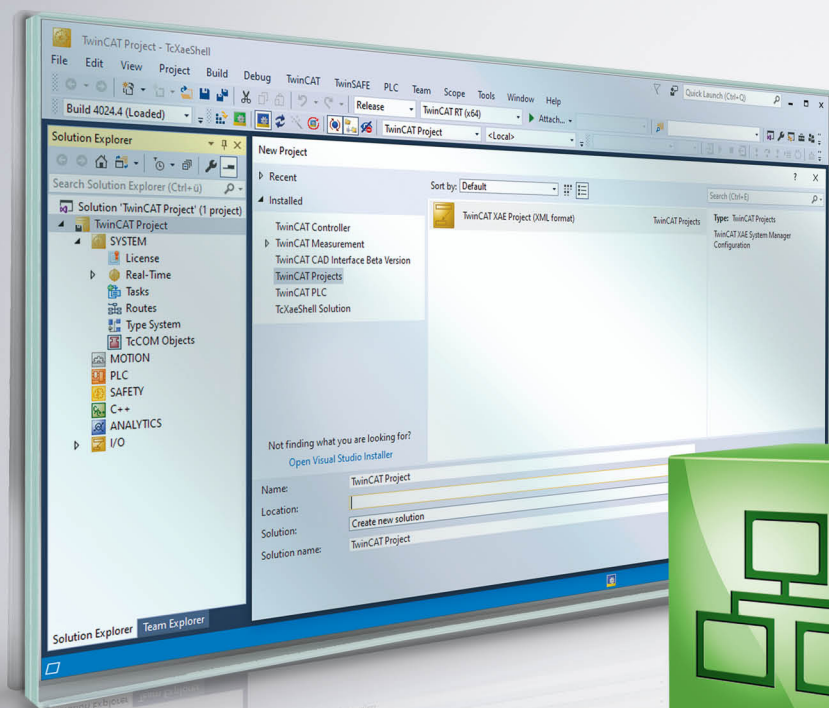


Table of contents

1 Foreword	5
1.1 Notes on the documentation	5
1.2 Safety instructions	6
1.3 Notes on information security	7
2 Overview	8
3 Installation	9
3.1 System requirements	9
3.2 Installation	9
3.3 Licensing	9
4 Technical introduction	12
4.1 Application samples	12
4.1.1 AWS IoT Core	12
4.1.2 Bosch IoT Suite	13
4.1.3 Google IoT Core	13
4.1.4 IBM Watson IoT	13
4.1.5 MathWorks ThingSpeak	14
4.1.6 Microsoft Azure IoT Hub	15
4.1.7 Sending SMS and E-Mail notifications	15
4.2 MQTT	23
4.3 Exponential backoff	27
4.4 JSON	28
4.5 JSON Web Token (JWT)	28
4.6 Security	28
4.6.1 Transport layer	29
4.6.2 Application level	33
4.7 Re-parameterization	33
4.8 I/O device	33
4.9 LastWill handling	39
5 PLC API	40
5.1 Tc3_lotBase	40
5.1.1 FB_lotMqttClient	40
5.1.2 ST_lotMqttWill	46
5.1.3 ST_lotMqttTLS	47
5.1.4 Message Queue	48
5.2 Tc3_JsonXml	51
5.2.1 Function blocks	51
5.2.2 Interfaces	128
6 Samples	134
6.1 lotMqttSampleUsingQueue	135
6.2 lotMqttSampleUsingCallback	137
6.3 lotMqttSampleTlsPsk	139
6.4 lotMqttSampleTlsCa	140
6.5 lotMqttSampleAwsIoT	140

6.6	lotMqttSampleAzureIoT Hub	142
6.7	lotMqttSampleBoschIoT	144
6.8	lotMqttSampleGoogleIoT	145
6.9	lotMqttSampleIbmWatsonIoT	146
6.10	lotMqttSampleMathworksThingspeak	147
6.11	lotMqttSampleAzureIoT Hub DeviceTwin	148
6.12	JsonXmlSamples	148
6.12.1	Tc3JsonXmlSampleXmlDomWriter	148
6.12.2	Tc3JsonXmlSampleXmlDomReader	149
6.12.3	Tc3JsonXmlSampleJsonDomReader	151
6.12.4	Tc3JsonXmlSampleJsonSaxWriter	152
6.12.5	Tc3JsonXmlSampleJsonSaxReader	152
6.12.6	Tc3JsonXmlSampleJsonDataType	153
7	Appendix	156
7.1	Error Codes	156
7.2	ADS Return Codes	156
7.3	Error diagnosis	161
7.4	Support and Service	163
7.5	Cipher suites used	165

1 Foreword

1.1 Notes on the documentation

This description is only intended for the use of trained specialists in control and automation engineering who are familiar with applicable national standards.

It is essential that the documentation and the following notes and explanations are followed when installing and commissioning the components.

It is the duty of the technical personnel to use the documentation published at the respective time of each installation and commissioning.

The responsible staff must ensure that the application or use of the products described satisfy all the requirements for safety, including all the relevant laws, regulations, guidelines and standards.

Disclaimer

The documentation has been prepared with care. The products described are, however, constantly under development.

We reserve the right to revise and change the documentation at any time and without prior announcement. No claims for the modification of products that have already been supplied may be made on the basis of the data, diagrams and descriptions in this documentation.

Trademarks

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® and XPlanar® are registered trademarks of and licensed by Beckhoff Automation GmbH.

Other designations used in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owners.

Patent Pending

The EtherCAT Technology is covered, including but not limited to the following patent applications and patents:

EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
with corresponding applications or registrations in various other countries.



EtherCAT® is a registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany

Copyright

© Beckhoff Automation GmbH & Co. KG, Germany.

The reproduction, distribution and utilization of this document as well as the communication of its contents to others without express authorization are prohibited.

Offenders will be held liable for the payment of damages. All rights reserved in the event of the grant of a patent, utility model or design.

1.2 Safety instructions

Safety regulations

Please note the following safety instructions and explanations!

Product-specific safety instructions can be found on following pages or in the areas mounting, wiring, commissioning etc.

Exclusion of liability

All the components are supplied in particular hardware and software configurations appropriate for the application. Modifications to hardware or software configurations other than those described in the documentation are not permitted, and nullify the liability of Beckhoff Automation GmbH & Co. KG.

Personnel qualification

This description is only intended for trained specialists in control, automation and drive engineering who are familiar with the applicable national standards.

Description of symbols

In this documentation the following symbols are used with an accompanying safety instruction or note. The safety instructions must be read carefully and followed without fail!

DANGER

Serious risk of injury!

Failure to follow the safety instructions associated with this symbol directly endangers the life and health of persons.

WARNING

Risk of injury!

Failure to follow the safety instructions associated with this symbol endangers the life and health of persons.

CAUTION

Personal injuries!

Failure to follow the safety instructions associated with this symbol can lead to injuries to persons.

NOTE

Damage to the environment or devices

Failure to follow the instructions associated with this symbol can lead to damage to the environment or equipment.



Tip or pointer

This symbol indicates information that contributes to better understanding.

1.3 Notes on information security

The products of Beckhoff Automation GmbH & Co. KG (Beckhoff), insofar as they can be accessed online, are equipped with security functions that support the secure operation of plants, systems, machines and networks. Despite the security functions, the creation, implementation and constant updating of a holistic security concept for the operation are necessary to protect the respective plant, system, machine and networks against cyber threats. The products sold by Beckhoff are only part of the overall security concept. The customer is responsible for preventing unauthorized access by third parties to its equipment, systems, machines and networks. The latter should be connected to the corporate network or the Internet only if appropriate protective measures have been set up.

In addition, the recommendations from Beckhoff regarding appropriate protective measures should be observed. Further information regarding information security and industrial security can be found in our <https://www.beckhoff.com/secguide>.

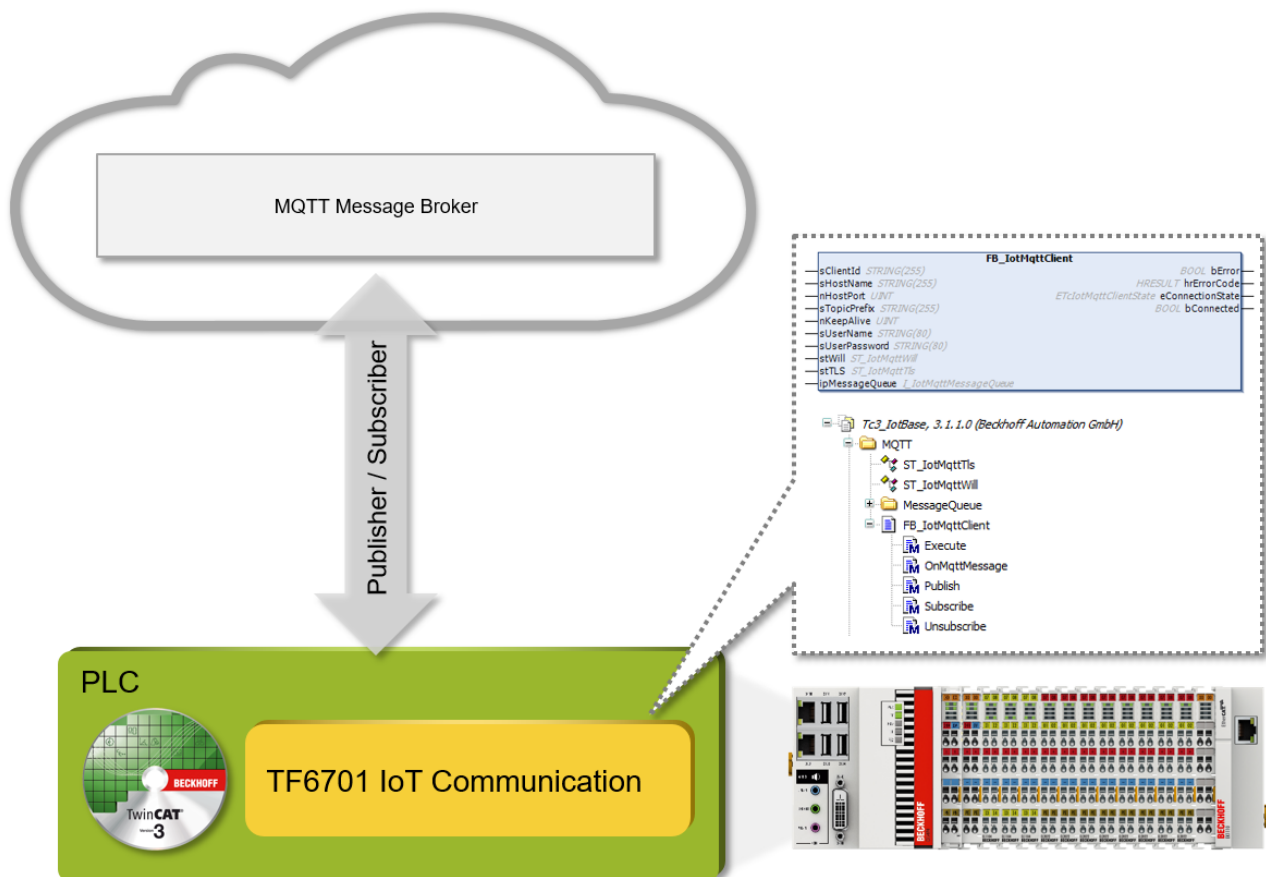
Beckhoff products and solutions undergo continuous further development. This also applies to security functions. In light of this continuous further development, Beckhoff expressly recommends that the products are kept up to date at all times and that updates are installed for the products once they have been made available. Using outdated or unsupported product versions can increase the risk of cyber threats.

To stay informed about information security for Beckhoff products, subscribe to the RSS feed at <https://www.beckhoff.com/secinfo>.

2 Overview

The function blocks of the PLC library Tc3_IotBase can be used for publisher/subscriber based data exchange between the TwinCAT PLC and a message broker via the MQTT communication protocol. Symbols can be sent (publish mode) and received (subscribe mode). The data format to be used is freely definable and can be created via additional PLC libraries, e.g. the Tc3_JsonXml library.

In addition to the PLC library, a TwinCAT I/O device is also available, via which a device-to-device communication connection can be configured via a message broker. The main application case for this I/O device is where two TwinCAT-based systems are to exchange data with each other via a message broker. The data format is by definition a binary format and not freely definable.



Product components

The function TF6701 IoT Communication consists of the following components, which are supplied with TwinCAT 3 as standard:

- **Drivers:** TcIotDrivers.sys (supplied with TwinCAT 3 XAE and XAR setups)
- **PLC library:** Tc3_IotBase (supplied with TwinCAT 3 XAE setup)

3 Installation

3.1 System requirements

Technical data	Description
Operating system	Windows 7/10, Windows Embedded Standard 7, Windows CE 7
Target platform	PC architecture (x86, x64 or ARM)
TwinCAT version	TwinCAT 3.1 build 4022.0 or higher
Required TwinCAT setup level	TwinCAT 3 XAE, XAR
Required TwinCAT license	TF6701 TC3 IoT Communication

3.2 Installation

A separate setup is not required for TF6701 IoT Communication. All required components will be delivered directly within the TwinCAT setup.

- TwinCAT XAE setup: Driver components and PLC library (Tc3_lotBase)
- TwinCAT XAR setup: Driver components

3.3 Licensing

The TwinCAT 3 function can be activated as a full version or as a 7-day test version. Both license types can be activated via the TwinCAT 3 development environment (XAE).

Licensing the full version of a TwinCAT 3 Function

A description of the procedure to license a full version can be found in the Beckhoff Information System in the documentation "[TwinCAT 3 Licensing](#)".

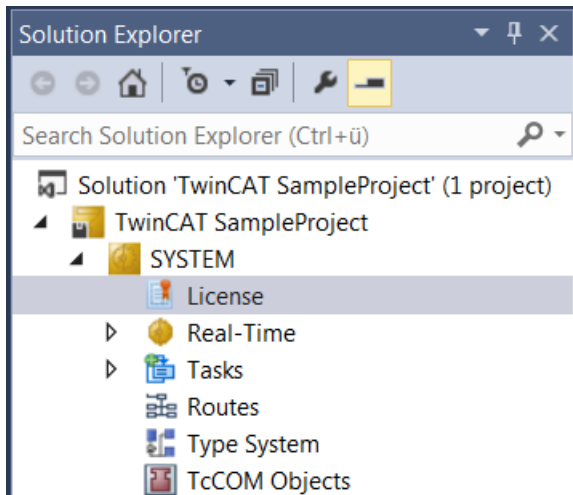
Licensing the 7-day test version of a TwinCAT 3 Function



A 7-day test version cannot be enabled for a TwinCAT 3 license dongle.

1. Start the TwinCAT 3 development environment (XAE).
2. Open an existing TwinCAT 3 project or create a new project.
3. If you want to activate the license for a remote device, set the desired target system. To do this, select the target system from the **Choose Target System** drop-down list in the toolbar.
 - ⇒ The licensing settings always refer to the selected target system. When the project is activated on the target system, the corresponding TwinCAT 3 licenses are automatically copied to this system.

4. In the **Solution Explorer**, double-click **License** in the **SYSTEM** subtree.



⇒ The TwinCAT 3 license manager opens.

5. Open the **Manage Licenses** tab. In the **Add License** column, check the check box for the license you want to add to your project (e.g. "TF4100 TC3 Controller Toolbox").

Order Information (Runtime) **Manage Licenses** Project Licenses Online Licenses

☐ Disable automatic detection of required licenses for project

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

6. Open the **Order Information (Runtime)** tab.

⇒ In the tabular overview of licenses, the previously selected license is displayed with the status "missing".

7. Click **7-Day Trial License...** to activate the 7-day trial license.

The screenshot shows the 'Order Information (Runtime)' tab in the Beckhoff license management software. It contains three main sections: 'License Device', 'License Request', and 'License Activation'. The 'License Activation' section at the bottom is highlighted with a red rectangular box. Inside this section, the '7 Days Trial License...' button is also highlighted with a red rectangular box. Other visible elements include the 'License Device' section with a dropdown menu set to 'Target (Hardware Id)' and an 'Add...' button, and the 'License Request' section with a 'Provider' dropdown set to 'Beckhoff Automation' and a 'Generate File...' button.

- ⇒ A dialog box opens, prompting you to enter the security code displayed in the dialog.

The screenshot shows a dialog box titled 'Enter Security Code'. It prompts the user to 'Please type the following 5 characters:'. Below this prompt, the security code 'Kg8T4' is displayed in a box. To the right of the code, the 'OK' button is highlighted with a red rectangular box. Below the code box, there is an empty input field for the user to type the code. At the bottom right of the dialog, there is a 'Cancel' button.

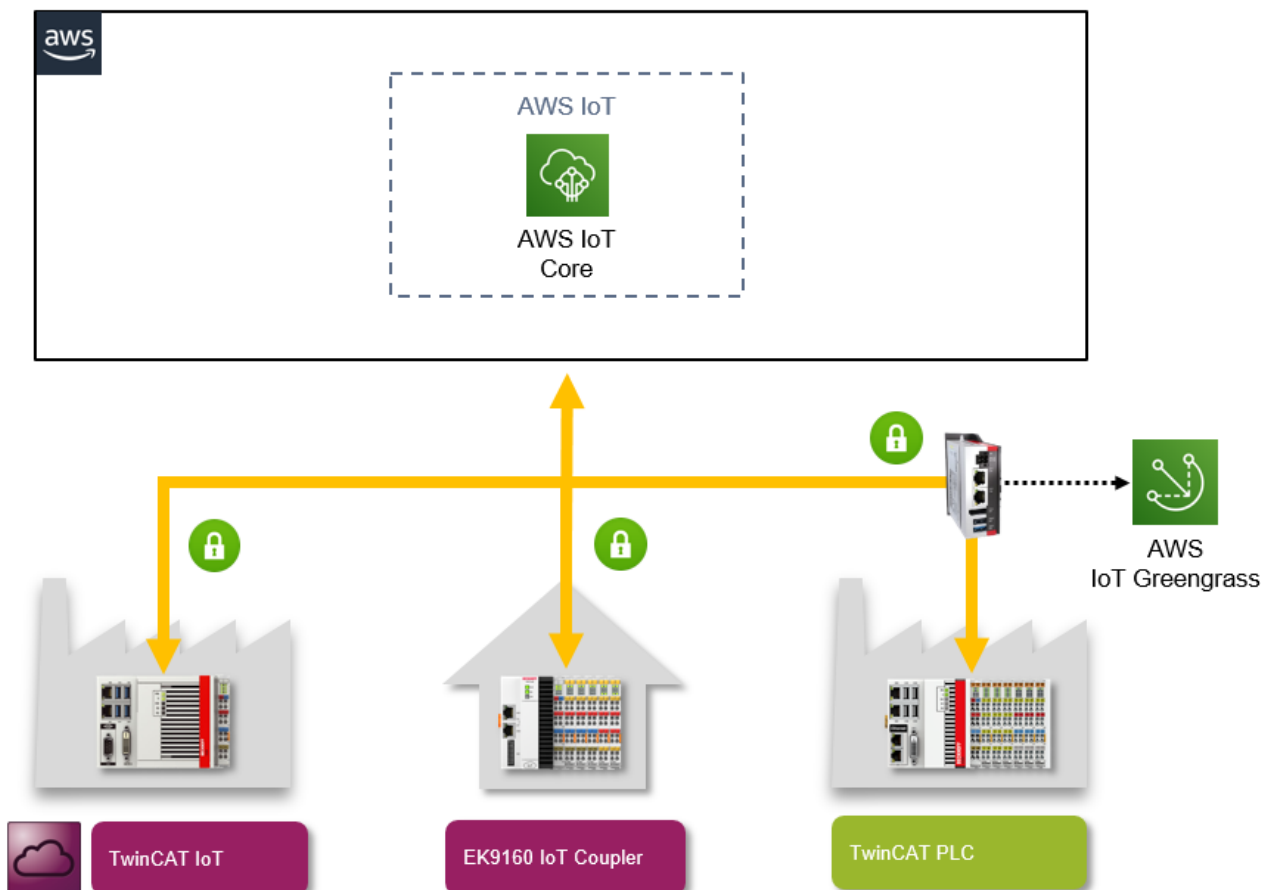
8. Enter the code exactly as it is displayed and confirm the entry.
9. Confirm the subsequent dialog, which indicates the successful activation.
- ⇒ In the tabular overview of licenses, the license status now indicates the expiry date of the license.
10. Restart the TwinCAT system.
- ⇒ The 7-day trial version is enabled.

4 Technical introduction

4.1 Application samples

4.1.1 AWS IoT Core

AWS IoT Core is a managed cloud service that enables connected devices to easily and securely communicate bi-directionally with cloud applications and other devices. Special functionalities, such as the AWS IoT Device Shadow Service, enable communication with devices that are not yet connected.



AWS IoT Core and TwinCAT IoT

AWS IoT Core is based on the MQTT transport protocol. It is therefore possible to use TwinCAT IoT to send or receive messages to or from AWS IoT Core or AWS IoT Greengrass.

Various samples illustrate how to connect to the AWS IoT Core service.

Sample	Product	Description
lotMqttSampleAwsIoT [► 140]	TF6701	This sample demonstrates how you can use MQTT function blocks within the PLC logic to connect to AWS IoT Core and exchange data.
Data Agent AWS IoT	TF6720	This sample demonstrates how to configure the TwinCAT IoT Data Agent to connect to AWS IoT Core and exchange data.

In a similar way, the samples can also be applied to AWS IoT Greengrass. The associated AWS IoT Greengrass Core can be installed and operated on a C6015 Industrial PC, for example. Typically, the Greengrass Core (or a Lambda function provided there) initiates data communication with the controller, e.g.

via OPC UA. However, since the Greengrass Core has an integrated message broker, this communication direction may be reversed. In this case, TwinCAT IoT would establish a connection to the Greengrass Core via an MQTT channel and exchange data accordingly, for example.



Further Information

Further information about AWS IoT Core can be found in the [AWS IoT Core documentation](#).

4.1.2 Bosch IoT Suite

The Bosch IoT Suite is an IoT platform based on open standards and open source and supports seamless integration into the Bosch IoT eco-system. The Bosch IoT Hub provides devices with various communication interfaces for exchanging data with the Bosch IoT Cloud.

Bosch IoT Suite and TwinCAT IoT

The Bosch IoT Suite includes an MQTT message broker referred to as the Bosch IoT Hub. It is therefore possible to use TwinCAT IoT to send or receive messages to or from the Bosch IoT Suite.

The following sample illustrates how to establish a connection to the Bosch IoT Hub.

Sample	Product	Description
lotMqttSampleBoschIoT [► 144]	TF6701	This sample demonstrates how you can use MQTT function blocks within the PLC logic to connect to the Bosch IoT Hub and exchange data.



Further Information

For more information about the Bosch IoT Suite please visit the [official Bosch IoT Suite website](#).

4.1.3 Google IoT Core

Google IoT Core is a managed cloud service that enables distributed devices to exchange data with the Google cloud through a secure transport channel.

Google IoT Core and TwinCAT IoT

Google IoT Core includes an MQTT message broker. It is therefore possible to use TwinCAT IoT to send or receive messages to or from Google IoT Core.

The following sample illustrates how to connect to Google IoT Core.

Sample	Product	Description
lotMqttSampleGoogleIoT [► 145]	TF6701	This sample demonstrates how you can use MQTT function blocks from within PLC logic to connect to Google IoT Core and exchange data.

4.1.4 IBM Watson IoT

IBM Watson IoT is an IoT suite in the IBM cloud, which offers several services for connecting IoT devices to IBM Bluemix services, processing incoming messages or sending messages to the devices. From a device perspective, the functionalities of IBM Watson IoT enable simple and safe connection of IoT devices with IBM services by facilitating bidirectional communication between the devices and IBM Watson IoT.

IBM Watson IoT and TwinCAT IoT

Since IBM Watson IoT can be reached via the MQTT transport protocol, it is possible to use TwinCAT IoT for sending messages to IBM Watson IoT or receiving messages from it.

Various samples illustrate how to connect to IBM Watson IoT.

Sample	Product	Description
lotMqttSampleIbmWatsonIoT [► 146]	TF6701	This sample demonstrates how you can use MQTT function blocks to connect to IBM Watson IoT and exchange data.
Publication of data to IBM Watson IoT	TF6720	This sample demonstrates how to configure the TwinCAT IoT Data Agent to establish a connection to IBM Watson IoT and exchange data.

4.1.5 MathWorks ThingSpeak

ThingSpeak™ is an IoT platform from The MathWorks®, well known among other things for the software solutions MATLAB® and Simulink®.

The platform offers (apart from a REST API) an MQTT interface, via which the data from the TwinCAT runtime can be sent to ThingSpeak™. ThingSpeak™ enables the collection, storage, analysis, visualization of and reaction to incoming data. An important point that sets it apart from other platforms is the option to write MATLAB® code in the web browser, which can be used for the analysis and visualization of the data. It is also possible to use existing licenses for toolboxes from the On-premis programming environment in ThingSpeak™.

Functioning

The data ingest and the saving of data take place on the basis of so-called channels. Each channel has 8 fields that can be filled with incoming data. Apart from the 8 data fields there are further meta fields available such as latitude, longitude, altitude or a time stamp. Data published on a channel are stored in a database with the option of a data export (JSON, XML, CSV). The number of messages per time unit that can be sent to a channel depends on the stored ThingSpeak™ license. The MQTT interface is currently based on the sending of strings that are interpreted by the ThingSpeak channel.

Examples of possible actions on ThingSpeak™:

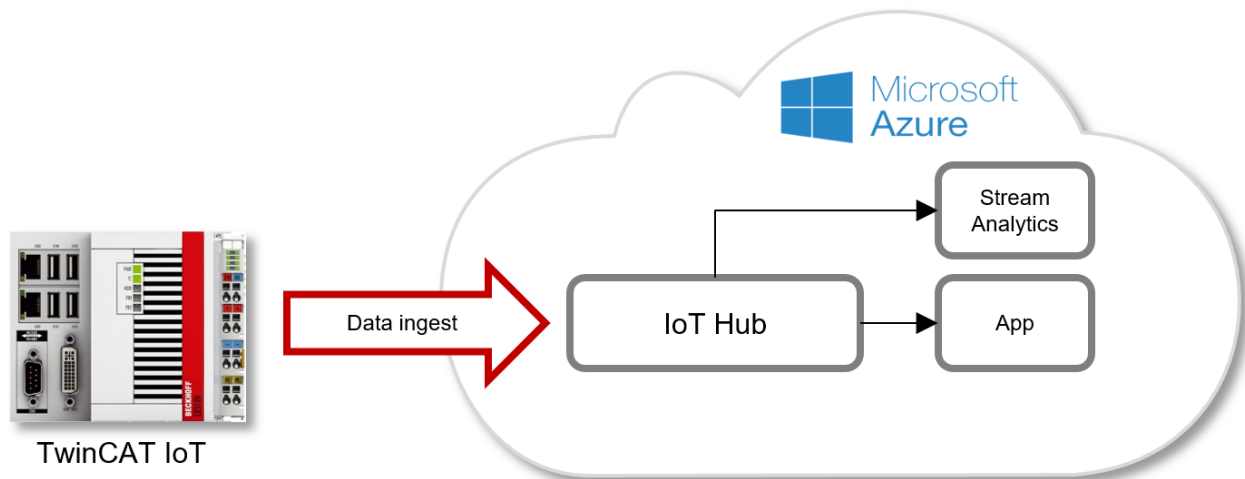
- Send a message to Twitter
- Cyclic execution of Matlab analysis scripts
- Execution of a Matlab analysis script, triggered by channel conditions.

Applications

On account of the data rate – limited by the cloud service – that can currently be sent by the controller to ThingSpeak™, a pronounced edge computing approach is a constructive strategy. MATLAB®/Simulink® models can be integrated in the TwinCAT runtime via the Beckhoff product TE1400 and algorithms for information densification in real time can be executed along with the various TwinCAT functions (Condition Monitoring, filter, etc.). Furthermore, processes with large time constants such as energy data management, building automation, etc. can be handled well with ThingSpeak™.

4.1.6 Microsoft Azure IoT Hub

The Microsoft Azure IoT Hub is an IoT suite in the Azure cloud, which offers several services for connecting IoT devices with Azure services, processing incoming messages or sending messages to the devices. From a device perspective, the functionalities of the Microsoft Azure IoT Hub enable simple and safe connection of IoT devices with Azure services by facilitating bidirectional communication between the devices and the Azure IoT Hub.



Microsoft Azure IoT Hub and TwinCAT IoT

The Microsoft Azure IoT Hub offers several communication interfaces for receiving or sending messages, including MQTT. It is therefore possible to use TwinCAT IoT to send messages to or receive messages from the Microsoft Azure IoT Hub or to control the Device Twin or execute method calls to the device.

Various samples illustrate how to connect to the Microsoft Azure IoT Hub.

Sample	Product	Description
lotMqttSampleAzureIoTHub [▶ 142]	TF6701	This sample demonstrates how you can use MQTT function blocks within the PLC logic to connect to the Microsoft Azure IoT Hub and exchange data.
Data Agent	TF6720	This sample demonstrates how to configure the TwinCAT IoT Data Agent to establish a connection to the Microsoft Azure IoT Hub and exchange data.
Data Agent Device Twin	TF6720	This sample demonstrates how to configure the TwinCAT IoT Data Agent to establish a connection to the Microsoft Azure IoT Hub and exchange data with the Device Twin.

4.1.7 Sending SMS and E-Mail notifications

4.1.7.1 Overview

For many years, customers have been sending machine status information and alarms via text messages and emails. Traditionally, either telephone dial-up (via a USB or serial modem) or an SMTP connection directly from the machine controller was used for this purpose. Although this kind of setup may have worked OK, the disadvantages of such an architecture are obvious:

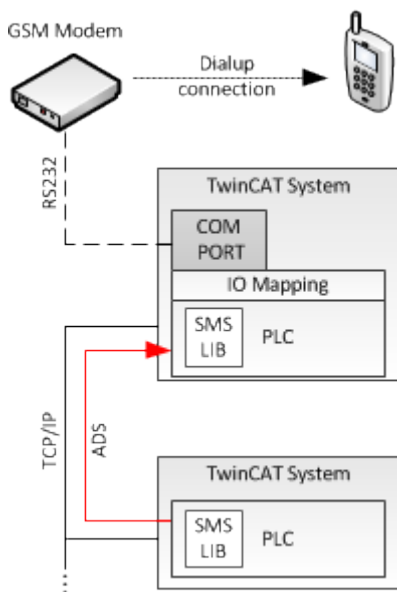
- for text messages: special modem hardware and a contract with a mobile phone provider are required

- for emails: an additional communication channel to a remote (mail) server is required

This documentation describes a more advanced, cloud-based approach to sending text and email messages. With this approach, the communication channel is derived from the actual message type (text, email), and the decision whether an incoming message from the machine is to be treated as text or email (or both) is made in the cloud.

4.1.7.2 Classic solution

The classic solution to send SMS and E-Mail notifications involves using our TwinCAT SMS/SMTP (TS6350, TF6350) supplement product. To send SMS notifications, the supplement product communicates with a GSM modem, which is attached via an RS232 serial connection to the controller. Beckhoff does not sell own GSM modems and a standard modem has to be bought on the market.



For E-Mail notifications, the supplement product establishes an SMTP connection to a mail server on the network. The mail server is then responsible for the message delivery.

4.1.7.3 Cloud-based solution

4.1.7.3.1 Requirements

Please ensure that the following requirements are met before you continue with this documentation.

- Make sure that you have created an AWS account and that you can access the AWS management console using the account credentials.

● Other TwinCAT IoT products and protocols

In this tutorial we will use TF6701 IoT Communication to connect to AWS IoT Core via MQTT. Please note that other products from the TwinCAT IoT product range can also connect to AWS IoT Core and that the same principles apply for these products. Also note that MQTT is not the only transport channel that can be used to connect to AWS IoT Core. Another transport option is HTTPS, which can be implemented with the TwinCAT IoT product TF6760 IoT HTTPS/REST.

- Install TwinCAT 3.1 Build 4022.0 or higher, so that the product TF6701 IoT Communication is available. We recommend updating to the latest TwinCAT version if possible.
- To understand how TwinCAT is linked to AWS IoT Core, please refer to the TF6701 documentation.

We particularly recommend the following articles about getting started with AWS. We will point out when it is important to read any of these articles before proceeding to the next step.

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

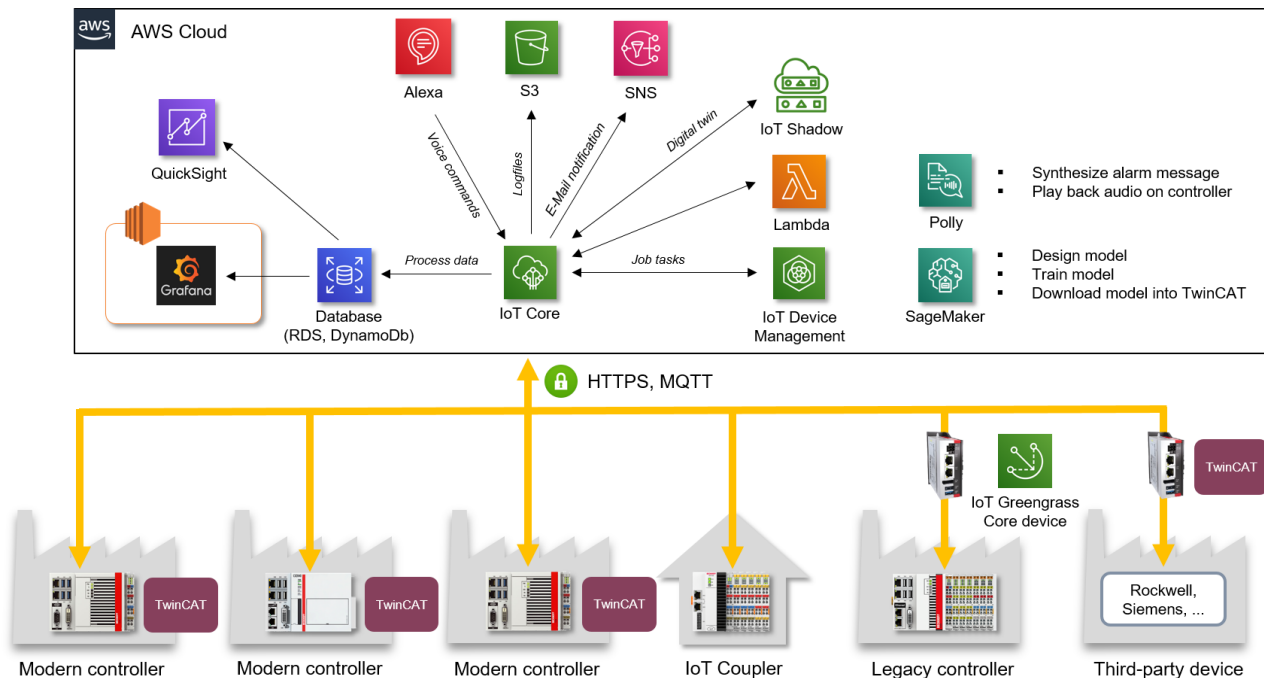
Amazon SNS: <https://docs.aws.amazon.com/sns/latest/dg/sns-getting-started.html>

Using Amazon SNS in an AWS IoT Core rule: <https://docs.aws.amazon.com/iot/latest/developerguide/config-and-test-rules.html>

4.1.7.3.2 Architecture

Cloud systems provide the technical infrastructure that makes the Internet of Things (IoT) possible. They also provide the scalability needed to support millions of connected devices. Cloud service providers like Microsoft Azure and Amazon Web Services (AWS) offer hundreds of services to support different use cases: virtual machines, message brokers, databases, serverless functions, etc. Their product portfolio also includes various functions for processing messages from connected devices. A central message broker (sometimes referred to as an IoT Hub) provides a single, secure endpoint for devices to communicate with other services in the ecosystem. Rules can be used to filter messages and forward them to other services. Because the message broker is treated as the single endpoint for all data connectivity scenarios, the firewall attack surface is minimized.

The following diagram illustrates this concept. It is based on services offered by AWS as an example. Other cloud platforms such as Microsoft Azure offer similar services.



Focusing on our use case (text messages and emails), the relevant services are:

- AWS IoT Core (message broker) for a single, secure endpoint in the cloud
- Amazon SNS for sending text messages and emails

Compared to a traditional setup, a cloud-based solution has the following advantages:

1. The message type (email, text, ...) is transparent for the device that issues the message. The device simply sends the message, while the decision whether it is sent in the form of an email or text message is made in the cloud.
2. Address changes do not have to be forwarded to the device that issues a message. All relevant contact data (email addresses, phone numbers, ...) are managed in the cloud.
3. Secure transport between the device that issues a message and the cloud. Each message of the device is sent to AWS IoT Core over a secure communication channel, either MQTT or HTTPS.
4. The required internet connectivity is based on TCP/IP. No additional modem hardware is required for sending text messages.

5. No contract with a mobile phone provider is required. AWS manages the text message transmission. Charges are based on usage (pay-per-use).

TwinCAT IoT supports MQTT and HTTPS connectivity with AWS IoT Core. The following pages of the documentation provide a more detailed configuration description for the components involved in this use case.

4.1.7.3.3 Setting up AWS IoT Core

AWS IoT Core is a scalable, managed message broker service in the AWS ecosystem. It facilitates secure device connection and data ingest management. To use AWS IoT Core, you need:

- An AWS account for signing in to the AWS web-based management console. All required AWS IoT Core features are automatically implemented when the account is created, so service provisioning is not necessary.
- Device credentials (certificates) and security policies for each connected device. The certificates must be transferred to the device and used by the device while connecting to the AWS IoT Core service. In other words, the certificates are always used by TwinCAT IoT when the connection is established. This is described in a later chapter.

The setup of AWS IoT Core includes the following topics:

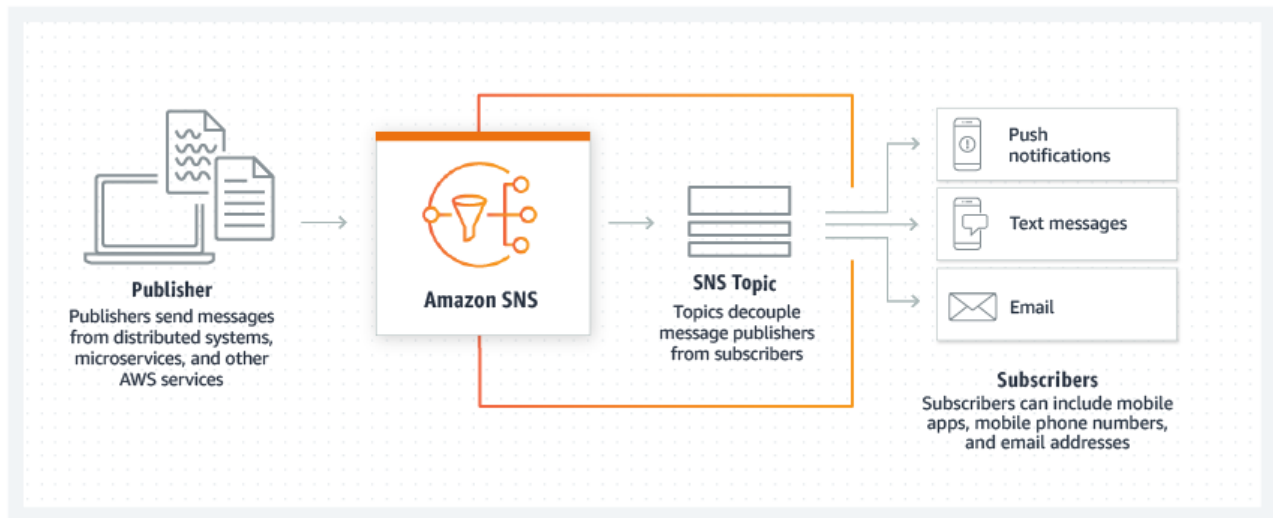
1. Logging into the AWS IoT console
2. Creating an object
3. Registering a device
4. Configuring your machine
5. Viewing the MQTT messages issued by the device with the AWS IoT MQTT client
6. Configuring and testing rules (next chapter)

These steps are described in detail in the tutorial [Getting Started with AWS IoT Core](#). This tutorial is a good source of information and describes the above steps in detail. We recommend that you read this tutorial and work through the step-by-step instructions before proceeding to the next chapter in this documentation.

Step 6 of the tutorial describes the exact use case we are trying to solve: Configuring an AWS IoT Core rule that uses Amazon SNS to send an email or text message.

4.1.7.3.4 Setting up Amazon SNS

Amazon SNS allows you to send push notifications to mobile apps, text messages to mobile numbers, and plain text emails to email addresses. [Step 6 \("Configuring and testing rules"\)](#) of the official [Getting Started with AWS IoT Core](#) guide describes in detail how Amazon SNS must be prepared for an AWS IoT rule to forward a device message to an SNS topic. If you want to learn more about Amazon SNS, we recommend the tutorial [Getting Started with Amazon SNS](#).

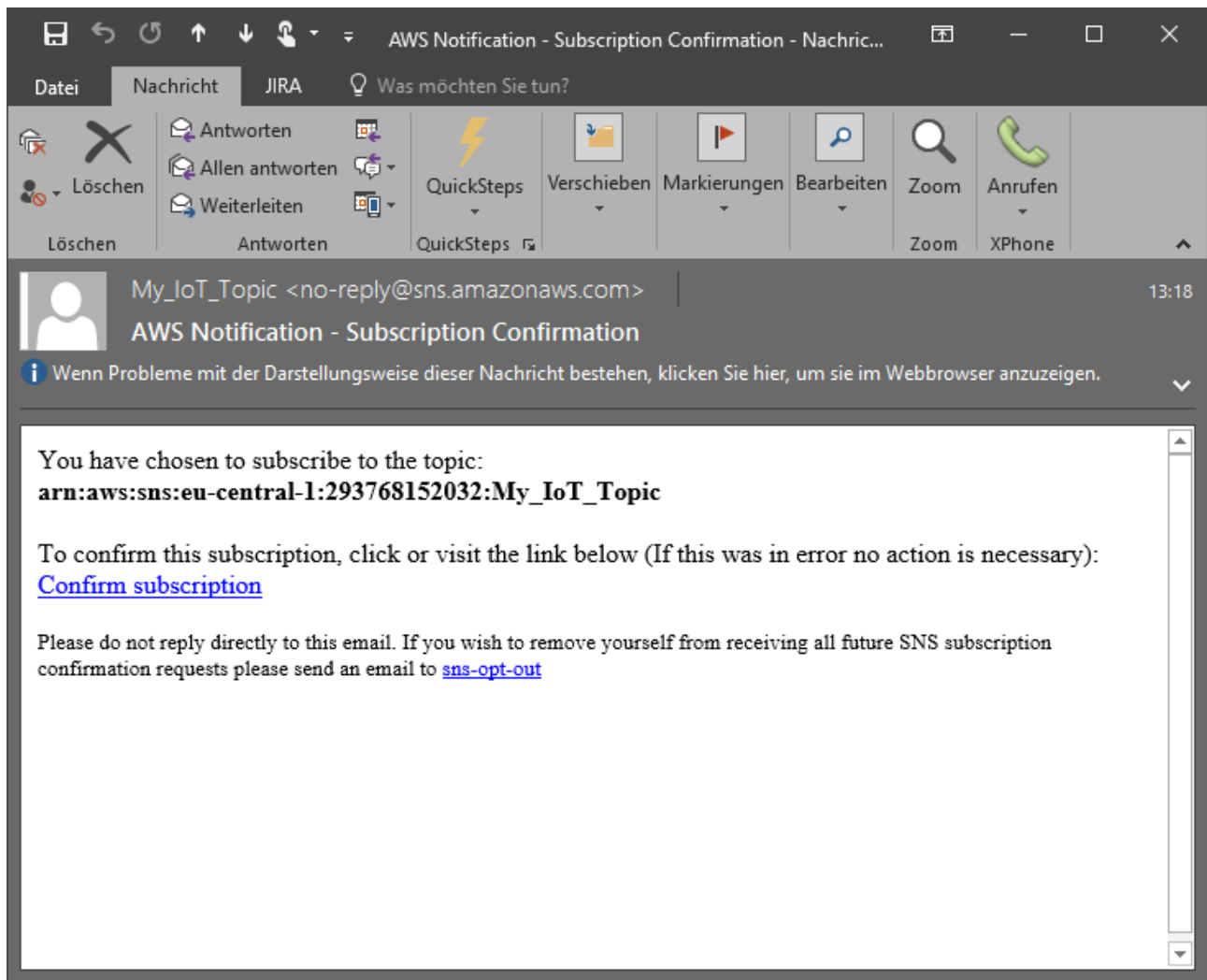


In this official graphic, AWS IoT Core acts as an intermediate station between the publisher of a message (the "device") and Amazon SNS.

Simply follow the steps described in the official Getting Started guide to

- create an Amazon SNS topic and a subscription (use email as "protocol")
- create an AWS IoT Core rule
- test the rule

Note the following: When using "email" as protocol, a one-off message will be sent to the email address entered by the user to confirm the subscription. This email must be acknowledged before messages can be sent to this email address. This procedure is similar to that for e-newsletters.

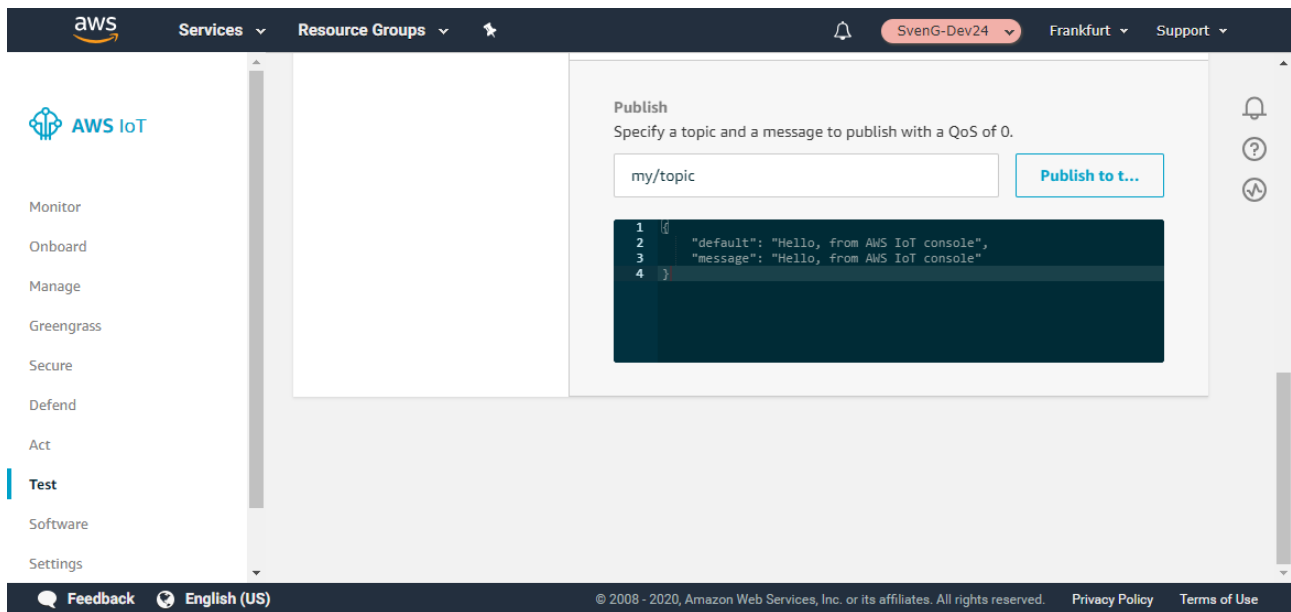


Note on service limits: Note that for regular AWS accounts there may be a limit for text messages. This limit can be increased by calling AWS support. For more information, visit the Amazon SNS service limit web page.

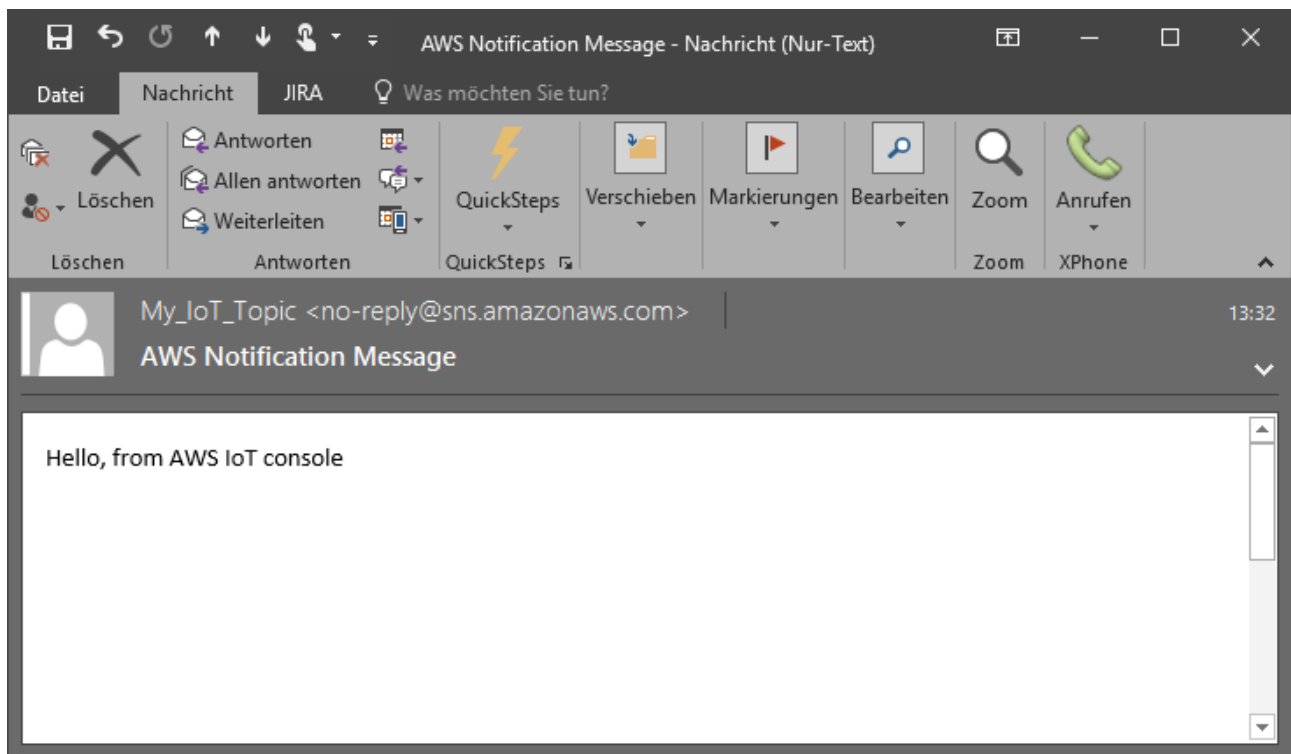
Once the Amazon SNS Topic/Subscription and the AWS IoT rule have been created, the setup can be tested using the MQTT client integrated into the AWS management console. To do this, simply send a test message to topic "my/topic" topic specified as a filter in the AWS IoT rule and use the following JSON content:

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

All [available properties](#) are documented in the Amazon SNS documentation.



The notification email should arrive at the email address used for the Amazon SNS subscription after a few seconds.



In the next step we want to enable TwinCAT to send messages to AWS IoT Core.

4.1.7.3.5 Setting up TwinCAT

The TwinCAT IoT Supplement products facilitate cloud connectivity for different use cases. One of their main advantages is that they use standard communication protocols to provide connectivity to cloud systems from different vendors, such as Microsoft Azure, Amazon Web Services, IBM, Google, etc.

In this documentation, we use TF6701 IoT Communication to connect to AWS IoT Core and publish a message for the topic "my/topic" that has been set as a filter in the AWS IoT rule to forward the messages that arrive at Amazon SNS for that particular topic, in order to send a notification email.

Requirements

This chapter is based on the regular TF6701 sample "IotMqttSampleAwsIoT", which illustrates the general procedure for connecting to AWS IoT Core. Download this sample to establish a common starting point. For more detailed information about how the sample code works, please refer to the corresponding Infosys website for this particular sample.



Important

Before you continue please ensure you have completed all the steps described in chapter [Setup of AWS IoT Core \[► 18\]](#).

Establishment of a connection

All certificates created with the AWS Management Console must be referenced in the FB_IotMqttClient.stTLS data structure (sCA, sCert and sKeyFile). Use the URL of the AWS IoT Core instance as the sHostName, as shown on the AWS Management Console. Since the connection is a secure MQTT connection, use 8883 as nHostPort. The MQTT client ID (sClientId) is the object name (ThingName) that was used when the object was created according to the chapter [Setup of AWS IoT Core \[► 18\]](#).

```
(* TLS settings for AWS IoT connection *)
fbMqttClient.stTLS.sCA := 'c:\certs\AmazonRootCA1.pem';
fbMqttClient.stTLS.sCert := 'c:\certs\6alba937cb-certificate.pem.crt';
fbMqttClient.stTLS.sKeyFile := 'c:\certs\6alba937cb-private.pem.key';

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

Defining appropriate topics

The standard sample illustrates how to connect to AWS IoT Core for exchange data with this message broker. It publishes messages for a topic and subscribes to a topic to receive messages. In the sample both topics are identical, so that TwinCAT receives the same message that it sent to the broker.

In addition to this regular sample behavior, we will now write new code to make the sample send a message for the topic "my/topic" so that an email is sent. For this purpose we first declare some new variables:

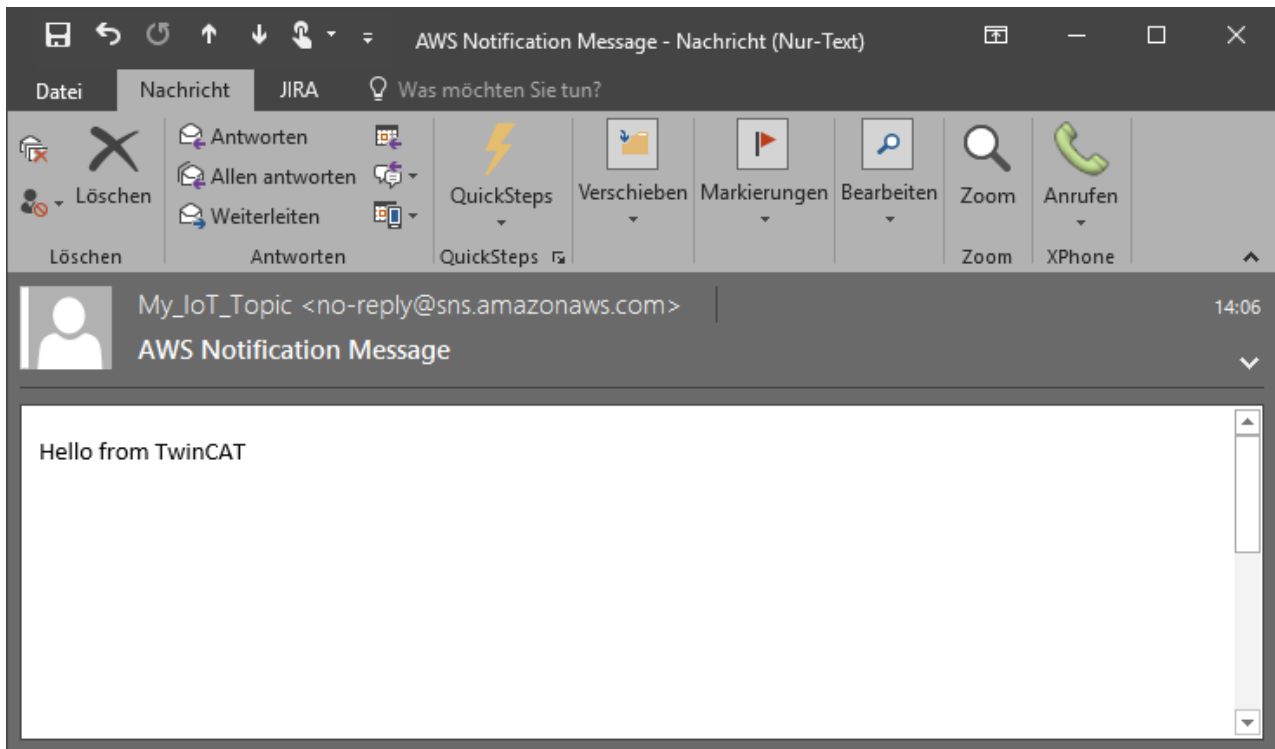
```
sTopicEmail : STRING(255) := 'my/topic';
bSendEmail : BOOL;
sPayloadEmail : STRING(255) := '{"default": "Hello from TwinCAT","message": "Hello from TwinCAT"}';
```

Then we will add the following lines of code after the IF query for the timer execution:

```
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
```

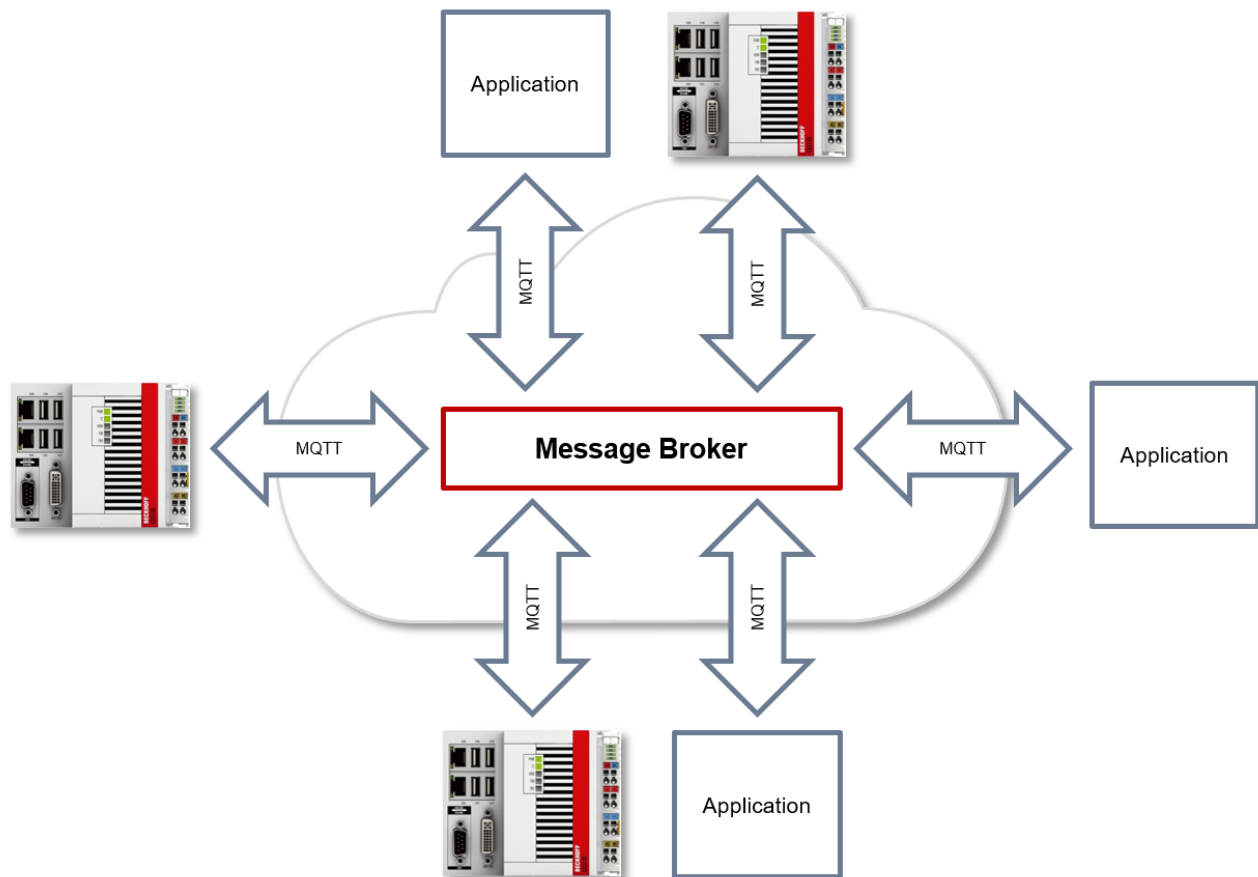
After activating the project, first validate whether the connection with AWS IoT Core was successful by checking the parameter eConnectionState (must be and remain "MQTT_ERR_SUCCESS"). If the connection status appears to fluctuate or if an TLS error is reported (e.g. "MQTT_ERR_TLS_VERIFICATIONFAILED"), double-check the steps described in chapter "Setup of AWS IoT Core" and make sure that the device certificate has been activated and the security policy allows the device to publish data for the topics used.

If the connection is successful, try setting bSendEmail to TRUE. After a few seconds, an email should appear in the inbox of the email address that was used for the Amazon SNS subscription.



4.2 MQTT

MQTT(Message Queuing Telemetry Transport) is a publisher/subscriber-based communication protocol, which enables message-based transfer between applications. A central component of this transfer type is the so-called message broker, which distributes messages between the individual applications or the sender and receiver of a message. The message broker decouples the sender and receiver, so that it is not necessary for the sender and receiver to know their respective address information. During sending and receiving all communication devices contact the message broker, which handles the distribution of the messages.



Payload

The content of an MQTT message is referred to as payload. Data of any type can be transferred, e.g. text, individual numerical values or a whole information structure.



Message payload formatting

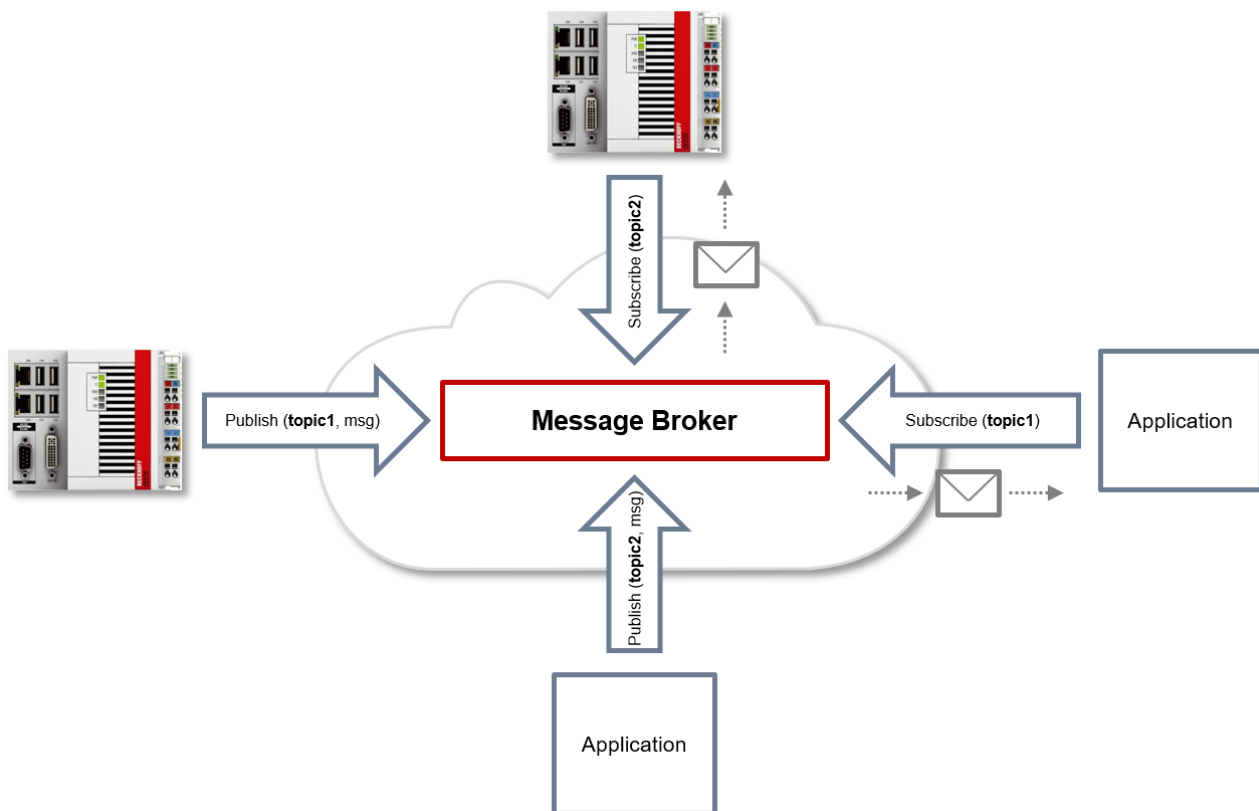
Note that the data type and the formatting of the content must be known to the sender and receiver side, particularly when binary information (alignment) or strings (with or without zero termination) are sent.

Topics

If a message broker is used that is based on the MQTT protocol, sending (publish mode) and subscribing (subscribe mode) of messages is organized with the aid of so-called topics. The message broker filters incoming messages based on these topics for each connected client. A topic may consist of several levels; the individual levels are separated by “/”.

Example: Campus / Building1 / Floor2 / Room3 / Temperature

When a publisher sends a message, it always specifies for which topic it is intended. A subscriber indicates which topic it is interested in. The message broker forwards the message accordingly.



Communication example 1 from the diagram above:

- An application subscribes to “topic1”.
- A controller publishes a message to “topic1”.
- The message broker forwards the message to the application accordingly.

Communication example 2 from the diagram above:

- A controller subscribes to “topic2”.
- An application publishes a message to “topic2”.
- The message broker forwards the message to the controller accordingly.

Wildcards

It is possible to use wildcards in conjunction with topics. A wildcard is used to represent part of the topic. In this case a subscriber may receive messages from several topics. A distinction is made between two types of wildcards:

- Single-level wildcards
- Multi-level wildcards

Example for single-level wildcard:

The + symbol describes a single-level wildcard. If it is used by the subscriber as described below, for example, corresponding messages to the topics are either received by the subscriber or not.

- The receiver subscribes to Campus/Building1/Floor2/+/Temperature
- The publisher sends to Campus/Building1/Floor2/Room1/Temperature - OK
- The publisher sends to Campus/Building1/Floor2/Room2/Temperature - OK
- The publisher sends to Campus/Building42/Floor1/Room1/Temperature - NOK
- The publisher sends to Campus/Building1/Floor2/Room1/Fridge/Temperature - NOK

Example for multi-level wildcard:

The # symbol describes a multi-level wildcard. If it is used by the subscriber as described below, for example, corresponding messages to the topics are either received by the subscriber or not. The # symbol must always be the last symbol in a topic string.

- The receiver subscribes to Campus/Building1/Floor2/#
- The publisher sends to Campus/Building1/Floor2/Room1/Temperature - OK
- The publisher sends to Campus/Building1/Floor2/Room2/Temperature - OK
- The publisher sends to Campus/Building42/Floor1/Room1/Temperature - NOK
- The publisher sends to Campus/Building1/Floor2/Room1/Fridge/Temperature - OK
- The publisher sends to Campus/Building1/Floor2/Room1/Humidity - OK

QoS (Quality of Service)

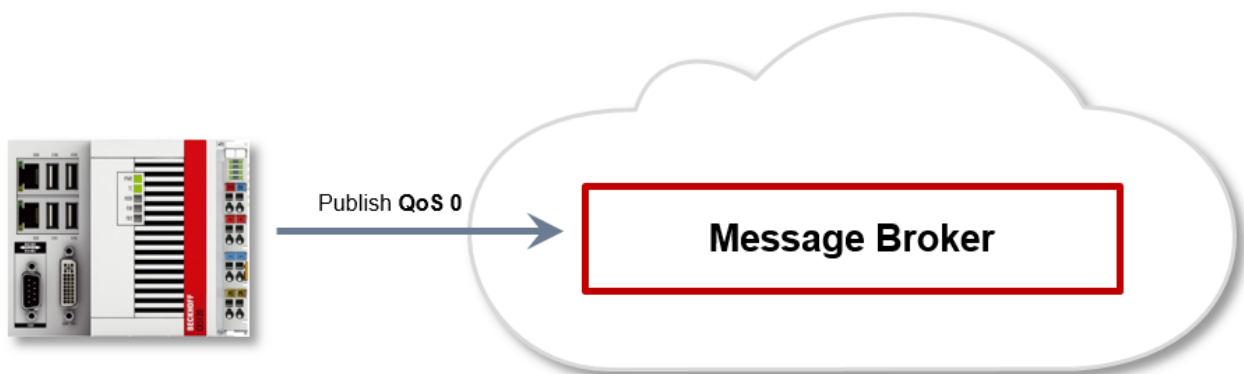
QoS is an arrangement between the sender and receiver of a message with regard to guaranteeing of the message transfer. MQTT features three different levels:

- 0 – not more than once
- 1 – at least once
- 2 – exactly once

Both types of communication (publish/subscribe) with the message broker must be taken into account and considered separately. The QoS level that a client uses for publishing a message is set by the respective client. When the broker forwards the message to client that has subscribed to the topic, the subscriber uses the QoS level that was specified when the subscription was established. This means that a QoS level that may have been specified as 2 by the publisher can be “overwritten” with 0 by the subscriber.

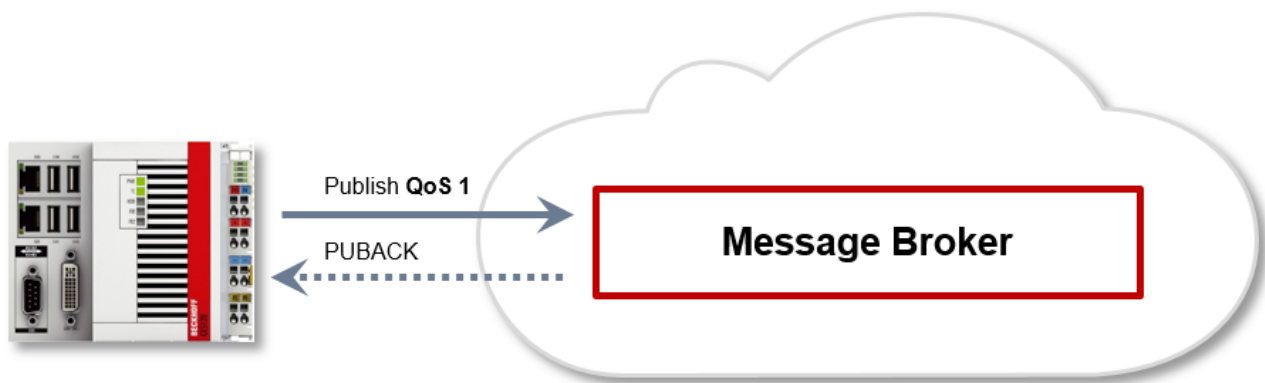
QoS-Level 0

At this QoS level the receiver does not acknowledge receipt. The message is not sent a second time.



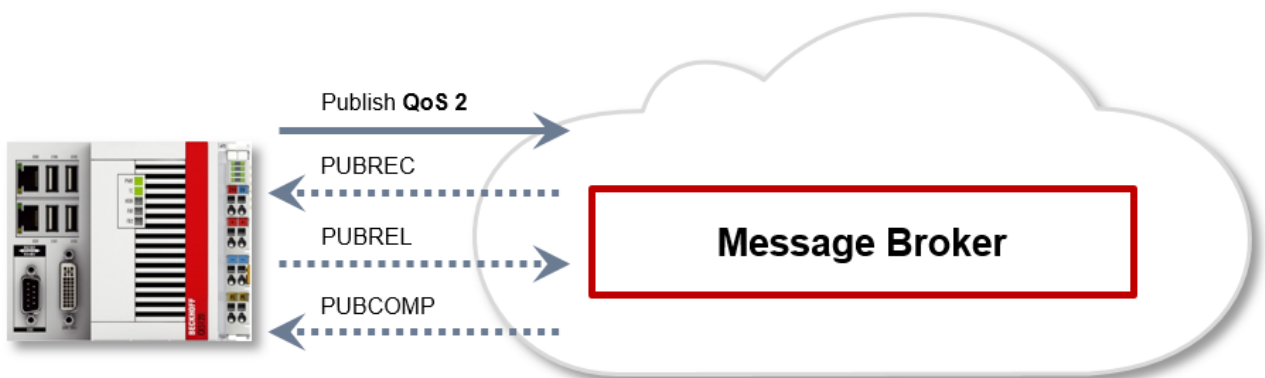
QoS-Level 1

At this QoS level the system guarantees that the message arrives at the receiver at least once, although the message may arrive more than once. The sender stores the message internally until it has received an acknowledgement from the receiver in the form of a PUBACK message. If the PUBACK message fails to arrive within a certain time, the message is resent.



QoS-Level 2

At this QoS level the system guarantees that the message arrives at the receiver no more than once. On the MQTT side this is realized through a handshake mechanism. QoS level 2 is the safest level (from a message transfer perspective), but also the slowest. When a receiver receives a message with QoS level 2, it acknowledges the message with a PUBREC. The sender of the message remembers it internally until it has received a PUBCOMP. This additional handshake (compared with QoS 1) is important for avoiding duplicate transfer of the message. Once the sender of the message receives a PUBREC, it can discard the initial publish information, since it knows that the message was received once by the receiver. In other words, it remembers the PUBREC internally and sends a PUBREL. Once the receiver has received a PUBREL, it can discard the previously remembered states and respond with a PUBCOMP, and vice versa. Whenever a package is lost, the respective communication device is responsible for resending the last message after a certain time.



Security

When a connection to the message broker is established, it is possible to use security mechanisms such as TLS, in order to encrypt the communication link or to realize authentication between client and message broker.

Sources

For further and more detailed information about MQTT we recommend the following blog series:

HiveMq blog: <http://www.hivemq.com/blog/mqtt-essentials/> (the main basis for this article)

4.3 Exponential backoff

A feature referred to as "exponential backoff" can be used to avoid burdening the message broker with unnecessary connection requests in case of a connection error. In the event of a TLS connection error involving the message broker, the reconnect rate is adjusted multiplicatively. This function can be activated using the `ActivateExponentialBackoff()` [► 46] method. The parameters of the method specify the minimum and maximum time for the algorithm. The minimum time describes the initial delay value for the new

connection attempt. The maximum time describes the highest delay value. The delay values are doubled until the maximum value is reached. Once a connection has been established, the backoff rate is reset to the original value. The [DeactivateExponentialBackoff\(\)](#) [► 46] method can be used to deactivate this function programmatically.

4.4 JSON

Most IoT services use the so-called JavaScript Object Notation (JSON) for describing data formats when transferring message contents. JSON is a slim data format in easily readable text form, in which data are organized in objects via property/value pairs.

Sample for a JSON object:

```
{
  "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"
    }
  }
}
```

The library Tc3_JsonXml, which is automatically installed with TwinCAT 3 XAE, facilitates creation and processing of JSON objects. (See documentation [PLC Lib: Tc3_JsonXml](#))

4.5 JSON Web Token (JWT)

JSON Web Token (JWT) is an open standard (based on RFC 7519) that defines a compact and self-describing format for securely transmitting information between communication devices in the form of a JSON object. The authenticity of the transmitted information can be verified and ensured, since a JWT is provided with a digital signature. The signature can involve a shared secret (via an HMAC algorithm) or a public/private key (via RSA).

The most common application example for JWT is the authorization of a device or user for a service. Once a user has logged into the service, all further requests to the service include the JWT. Based on the JWT, the service can then decide which additional services or resources the user may access. This means, for example, that single sign-on solutions can be implemented in cloud services.

The PLC library Tc3_JsonXml provides an option to create and sign a JWT via the method `FB_JwtEncode`.

4.6 Security

When considering protection of data communication, a distinction can be made between two levels: protection of the transport channel ([Transport layer](#) [► 29]) and protection at Application level. Both are described below.

4.6.1 Transport layer

The Standard Transport Layer Security (TLS) is used in the TwinCAT IoT driver for the secure transmission of data. **The following chapter describes the TLS communication flow for TLS version 1.2.** The TLS standard combines symmetric and asymmetric cryptography to protect transmitted data from unauthorized access and manipulation by third parties. In addition, TLS supports authentication of communication devices for mutual identity verification.

Contents of this chapter

The information in this chapter refers to the general TLS communication flow, without specific reference to the implementation in TwinCAT. They are only intended to provide a basic understanding in order to better comprehend the reference to the TwinCAT implementation explained in the following sub-chapters.

Cipher suite definition

A cipher suite as defined in TLS version 1.2 is a set of algorithms (key exchange, authentication, encryption, MAC) for encryption. The client and server agree on these during the TLS connection setup. For more information on cipher suites please refer to the relevant technical literature.

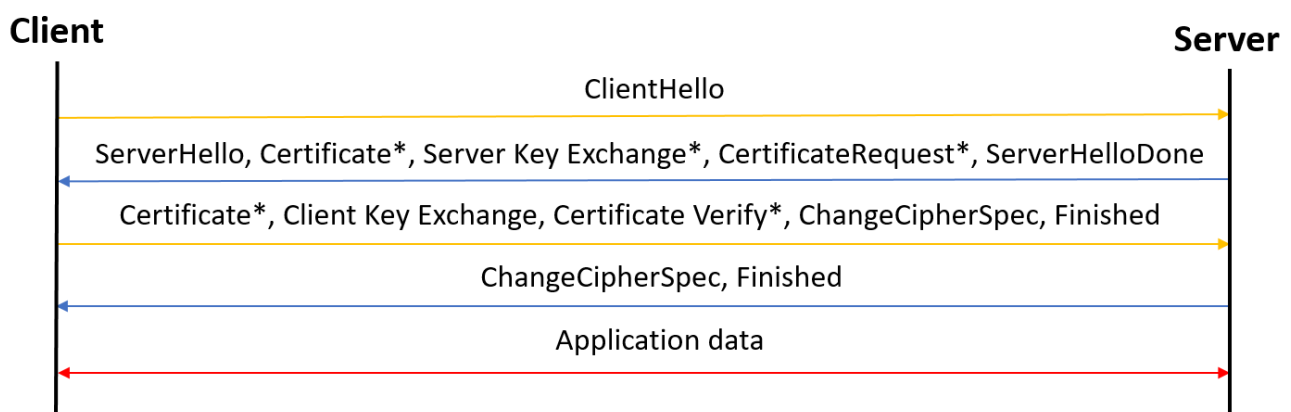
TLS communication flow

Communication with TLS encryption starts with a TLS handshake between server and client. During the handshake asymmetric cryptography is used; after successful completion of the handshake the server and client communicate based on symmetric cryptography, because this is many times faster than asymmetric cryptography.

There are three different types of authentication for the TLS protocol:

- The server identifies itself through a certificate (see [Server certificate](#) [► 30])
- Client and server identify themselves through a certificate (see [Client/Server certificate](#) [► 31])
- Pre-shared keys (see [Pre-shared keys](#) [► 32])

Please refer to the relevant technical literature for information about the advantages and disadvantages of the different authentication types.



Exemplary explanation based on RSA

All messages marked with * are optional, i.e. not mandatory. The following steps refer to the RSA procedure and are not generally valid for other procedures.

Client Hello: The client initiates a connection to the server. The TLS version used, a random sequence of bytes (client random) and the cipher suites supported by the client are transmitted, among other parameters.

Server Hello: The server selects one of the cipher suites offered by the client and specifies it for the communication. If there is no intersection between the cipher suites supported by the client and server, the TLS connection establishment is aborted. In addition, the server also communicates a random sequence of bytes (server random).

Certificate: The server presents its certificate to the client to enable the client to verify that the server is the expected server. If the client does not trust the server certificate, the TLS connection establishment is aborted. The server certificate also contains the server's public key.

(Server Key Exchange): For certain key exchange algorithms, the information from the certificate is not sufficient for the client to generate the so-called pre-master secret. In this case the missing information is transferred using Server Key Exchange.

Certificate Request: The server requests a certificate from the client to verify the identity of the client.

Server Hello Done: The server notifies the client that sending of the initial information is complete.

Certificate: The client communicates its certificate, including the public key, to the server. The procedure is the same as in the opposite direction: If the server does not trust the certificate sent by the client, the connection establishment is aborted.

Client Key Exchange: The client generates an encrypted pre-master secret and uses the server's public key to send the secret to the server using asymmetric encryption. This pre-master secret, the "server random" and the "client random" are then used to calculate the symmetric key that is used for communication after the connection has been established.

Certificate Verify: The client signs the previous handshake messages with its private key. Since the server has obtained the client's public key by sending the certificate, it can verify that the certificate presented really "belongs" to the client.

Change Cipher Spec: The client notifies the server that it is switching to symmetric cryptography. From here on every message from the client to the server is signed and encrypted.

Finished: The client notifies the server in encrypted form that the TLS connection establishment on its side is complete. The message contains a hash and a MAC relating the previous handshake messages.

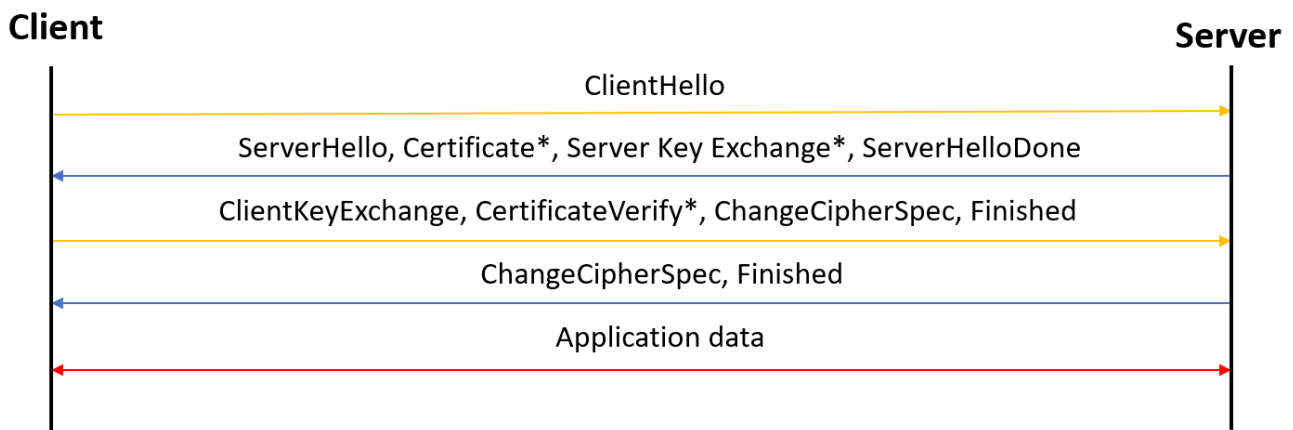
Change Cipher Spec: The server decrypts the pre-master secret that the client encrypted with its public key. Since only the server has its private key, only the server can decrypt this pre-master secret. This ensures that the symmetric key is only known to the client and the server. The server then calculates the symmetric key from the pre-master secret and the two random sequences and notifies the client that it too is now communicating using symmetric cryptography. From here on every message from the server to the client is signed and encrypted. By generating the symmetric key, the server can decrypt the client's Finished message and verify both hash and MAC. If this verification fails, the connection is aborted.

Finished: The server notifies the client that the TLS connection establishment on its side is also finished. As with the client, the message contains a hash and a MAC relating to the previous handshake messages. On the client side, the same verification is then performed as on the server. Again, if the hash and MAC are not successfully decrypted, the connection is aborted.

Application Data: Once the TLS connection establishment is complete, client and server communicate using symmetric cryptography.

4.6.1.1 Server certificate

This section covers a situation where the client wants to verify the server certificate but the server does not want to verify the client certificate. In this case the communication flow described in chapter [Transport layer](#) [► 29] is shortened as follows.



Verification of the server certificate

The server certificate is verified on the client side. A check is performed to ascertain whether it is signed by a particular certificate authority. If this is not the case, the client aborts the connection, since it does not trust the server.

Application in TwinCAT

In TwinCAT, the file path to the CA certificate is specified (.PEM or .DER file) or the content of the .PEM file as a string. The certificate presented by the server is then checked in the IoT driver. If the certificate chain is not signed by the specified CA, the connection to the server is aborted. The following code illustrates the described connection parameters as an example. The sample code refers to the HTTP client, although it is applicable to both the HTTP client and the MQTT client.

```

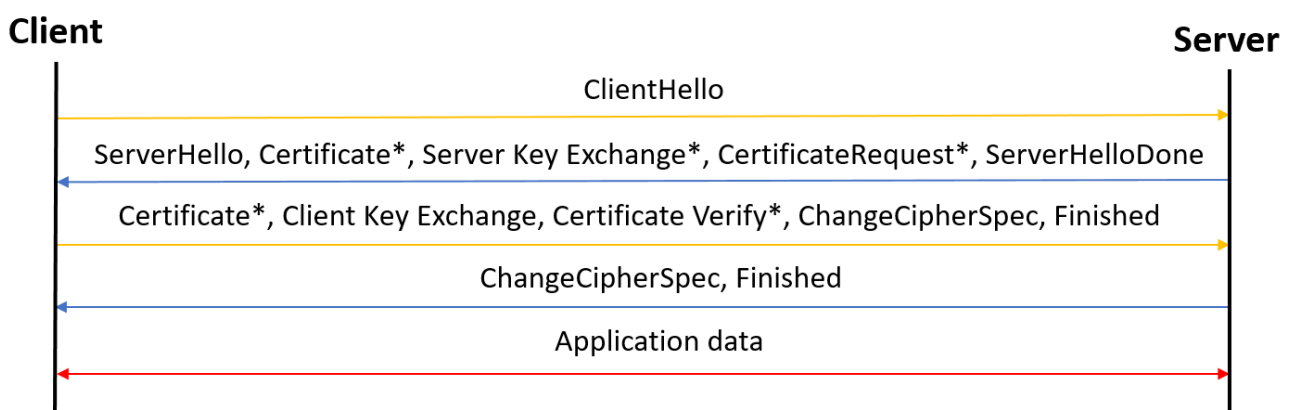
PROGRAM MAIN
VAR
    fbClient : FB_IotHttpClient;
END_VAR
fbClient.stTLS.sCA:= 'C:\TwinCAT\3.1\Config\Certificates\someCA.pem';
  
```

If the user does not have the CA certificate, a connection can still be established. A Boolean variable is available for this purpose, which prevents TwinCAT from verifying the server certificate. Although the connection is encrypted with the public key of the unverified server certificate, it is more vulnerable to man-in-the-middle attacks.

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

4.6.1.2 Client/Server certificate

This section considers the case where both the client certificate and the server certificate are verified. The slightly modified communication flow (compared to the [Server certificate](#) [► 30] chapter) is visualized in the following diagram. The individual steps of the TLS connection establishment are described in chapter [Transport layer](#) [► 29].



Application in TwinCAT

If a client certificate is used, in TwinCAT the file path (.PEM or .DER file) or the content of the .PEM file is passed as a string, just as for the CA certificate. TwinCAT as the client then presents this certificate to the server. For Certificate Verify the client's private key must also be referenced. Optionally, in the case of password protection for the private key, this password can also be transferred. The sample code refers to the HTTP client, although it is applicable to both the HTTP client and the MQTT client.

```
PROGRAM MAIN
VAR
    fbClient : FB_IotHttpClient;
END_VAR

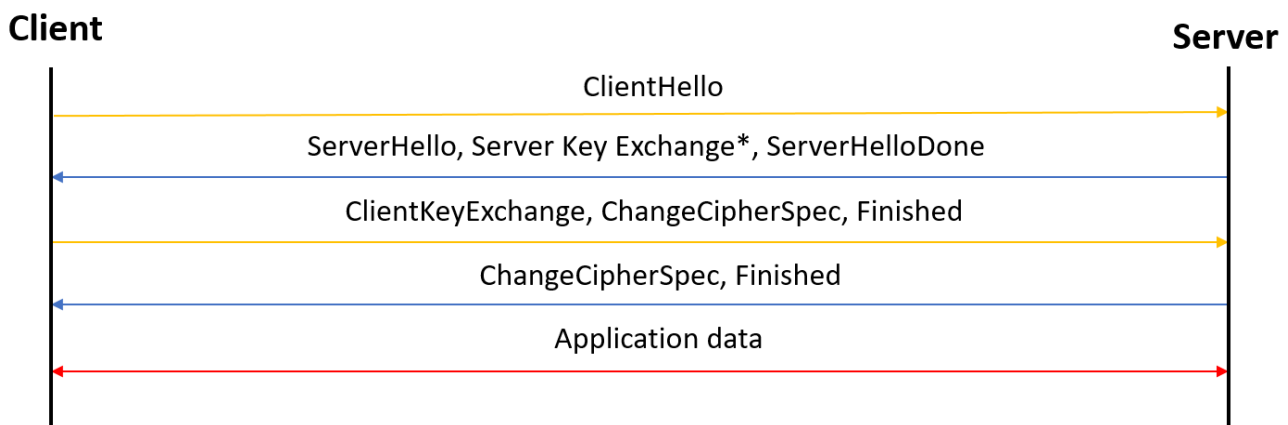
fbClient.stTLS.sCA:= 'C:\TwinCAT\3.1\Config\Certificates\someCA.pem';
fbClient.stTLS.sCert:= 'C:\TwinCAT\3.1\Config\Certificates\someCRT.pem';
fbClient.stTLS.sKeyFile:= 'C:\TwinCAT\3.1\Config\Certificates\someprivatekey.pem.key';
fbClient.stTLS.sKeyPwd:= 'yourkeyfilepasswordhere';
```

In principle, the user can also disable the validation of the server certificate in the case described. However, this is associated with weaker security (see [Server certificate](#) ► 301)

4.6.1.3 Pre-shared keys

By default, asymmetric key pairs are used for the TLS connection establishment. Asymmetric cryptography requires more computing power, so using pre-shared keys (PSK) may be an option in situations with limited CPU power. Pre-shared keys are previously shared symmetric keys.

Compared to the communication flow with asymmetric encryption, the certificate is omitted when using PSK. Client and server must agree on a PSK via the so-called identity. By definition the PSK is known in advance to both parties.



Server Key Exchange: In this optional message, the server can give the client a hint about the identity of the PSK used.

Client Key Exchange: The client specifies the identity of the PSK to be used for encryption.

Application in TwinCAT

In TwinCAT the identity of the PSK is specified as a string; the PSK itself is stored as a byte array in the controller. The length of the PSK is also specified. The sample code refers to the HTTP client, although it is applicable to both the HTTP client and the MQTT client.

```
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;
```

4.6.2 Application level

Various security mechanisms are also available at the application level. Several such security mechanisms are described below.

4.6.2.1 JSON Web Token (JWT)

JSON Web Token (JWT) is an open standard (based on RFC 7519) that defines a compact and self-describing format for securely transmitting information between communication devices in the form of a JSON object. The authenticity of the transmitted information can be verified and ensured, since a JWT is provided with a digital signature. The signature can involve a shared secret (via an HMAC algorithm) or a public/private key (via RSA).

The most common application example for JWT is the authorization of a device or user for a service. Once a user has logged into the service, all further requests to the service include the JWT. Based on the JWT, the service can then decide which additional services or resources the user may access. This means, for example, that single sign-on solutions can be implemented in cloud services.

The PLC library Tc3_JsonXml provides an option to create and sign a JWT via the method FB_JwtEncode.

4.7 Re-parameterization

In certain cases, it may be necessary to re-parameterize the MQTT client during operation by means of an online change. This new parameterization can either take place automatically in the program sequence or be triggered manually if required. This depends on the implementation.

Example use cases for a re-parameterization: replacement of a token that is about to expire, certificates that need to be renewed or a changed IP address of the MQTT broker. The following section describes the procedure for re-parameterization of a connected MQTT client.

In order for the TwinCAT MQTT client to be connected to a message broker, the [Execute \[► 42\]](#) method must be called cyclically in the background (see [FB lotMqttClient \[► 40\]](#)). When the program starts, the parameterization of the MQTT client instance is carried out first by calling this method, after which a connection to the broker is established. In order to carry out a re-parameterization, the call of the [Execute \[► 42\]](#) method must be negated, the parameters must then be reset and the call must then take place again as usual. The following code snippet shows a simple re-parameterization using a trigger.

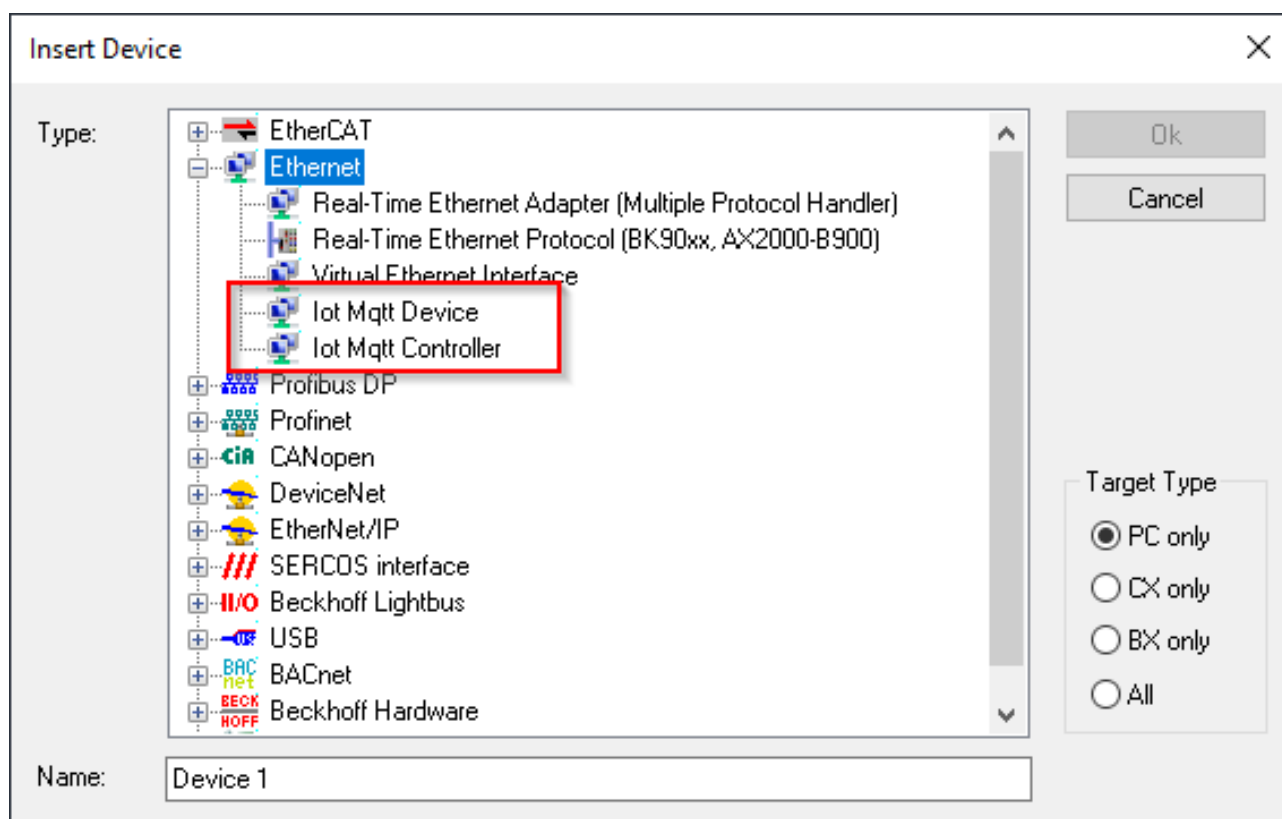
```
fbMqttClient.Execute(bConnect);

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

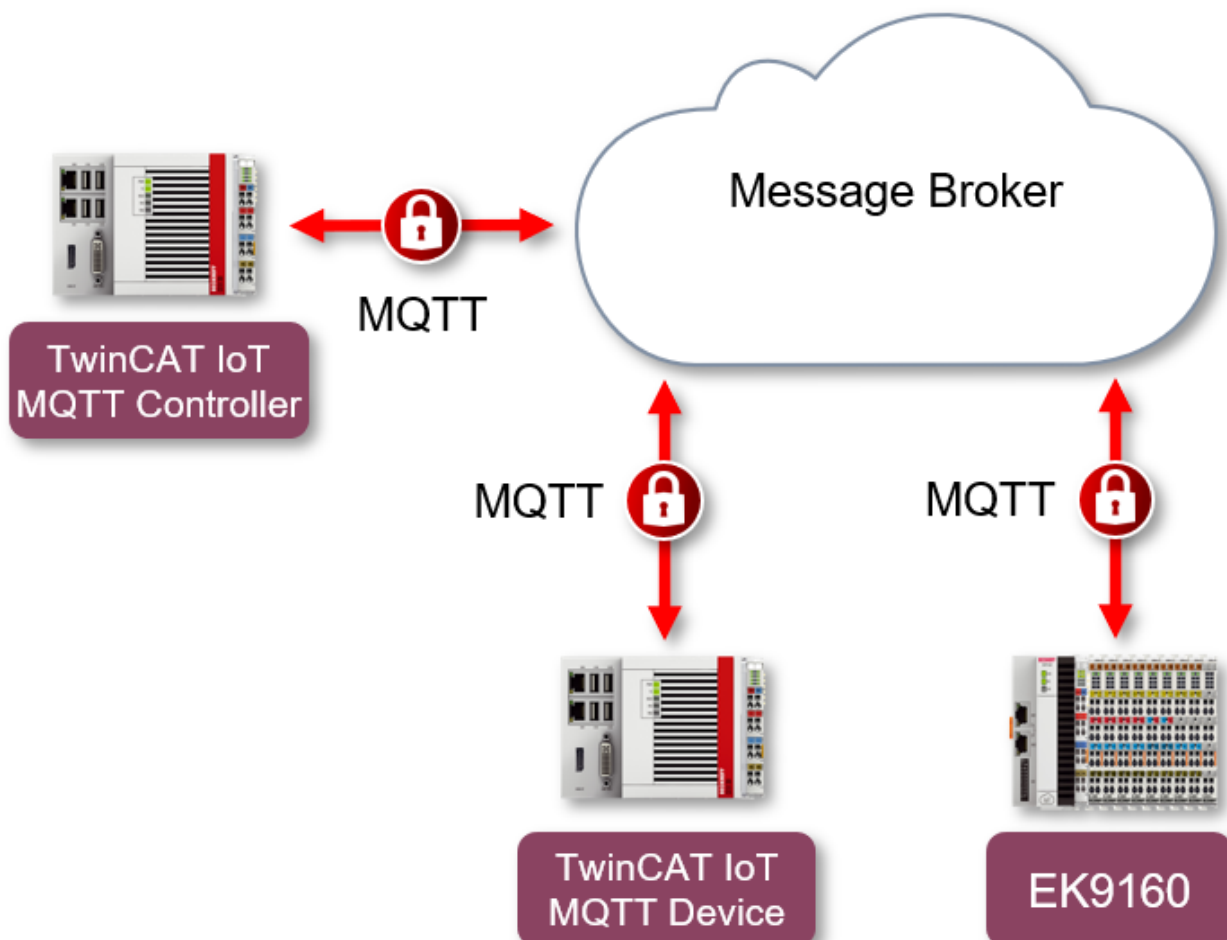
The call of the Execute method is negated by the trigger variable bReset, so that the parameters for the client ID and the host name can be reset afterwards. Once the trigger variable is set to FALSE again, the execute method is called cyclically again.

4.8 I/O device

In addition to the Tc3_lotBase PLC library, the IoT MQTT controller and IoT MQTT device provide two I/O devices that can be used to establish an MQTT-based communication link between two TwinCAT systems. Alternatively, an EK9160 IoT coupler can also be integrated into this type of communication.



The following diagram shows possible application scenarios. This type of communication link enables both TwinCAT systems to be coupled to each other and TwinCAT to be coupled to one or more EK9160s.



For an IoT MQTT Device, symbol information for all configured variables in the process image is stored on the message broker in a specific topic. An IoT MQTT controller then has the ability to scan this symbol information and create matching variables in its own process image. The EK9160 is automatically always an IoT MQTT device.

EK9160 configuration

The EK9160 is automatically configured as an IoT MQTT device in the background as soon as the device is configured to connect to an MQTT message broker. As a prerequisite, "Binary" must be selected as the data format and retain messages must be activated. The following screenshot shows the corresponding section of the EK9160 configuration web page.

Device 1

✓

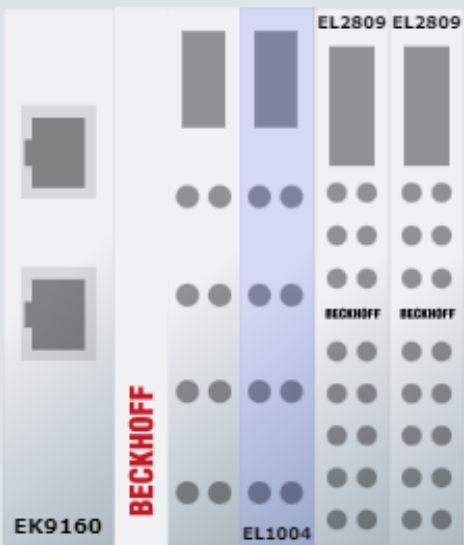
✕

Connection Type	General MQTT
MQTT Broker	172.17.98.43
Tcp Port	1883
ClientID	
Cycle Time (ms)	1000
Watchdog Mode	Disabled
Watchdog Timeout (ms)	5000
Retain	Allow retained messages
Data Format	Binary
Main Topic	EK9160
Publish Topic	EK9160/EK-58CE72/Stream1/Bin/Tx/Data
Subscribe Topic	EK9160/EK-58CE72/Stream1/Bin/Rx/Data
Username	
Password	
SAS-Token	
Connection Status	Connected
Publisher Send Count	4
Subscriber Receive Count	0
SSL/TLS-Mode	No Certificate

All I/O terminals have been activated for the communication link with the message broker. The following screenshot shows an example of this process on the configuration interface.

Configure I/O

Select Bus Terminal



Bus Terminal - EL1004

☒ ☐

Input	Publisher Symbol	Enabled <input checked="" type="checkbox"/>	Device
Channel 1			
Input	Slave 1 (EL1004).Channel 1.Input	<input checked="" type="checkbox"/>	Device 1 ▾
Channel 2			
Input	Slave 1 (EL1004).Channel 2.Input	<input checked="" type="checkbox"/>	Device 1 ▾
Channel 3			
Input	Slave 1 (EL1004).Channel 3.Input	<input checked="" type="checkbox"/>	Device 1 ▾
Channel 4			
Input	Slave 1 (EL1004).Channel 4.Input	<input checked="" type="checkbox"/>	Device 1 ▾

● TwinCAT as IoT MQTT device



In addition to the EK9160, TwinCAT itself can also act as an IoT MQTT device. In this case, the corresponding configuration steps must be carried out via the TwinCAT I/O area in TwinCAT XAE. The IoT MQTT device then behaves identically to the EK9160 with regard to the further process.

Configuration in TwinCAT

In order for TwinCAT to be able to process the symbol information and process values from the EK9160, an IoT MQTT controller must be created in the I/O area of TwinCAT XAE and configured for the connection with the message broker. It is important that the fields **Main Topic**, **Device** and **Stream** match the configuration of the EK9160. The following screenshot illustrates this process.

Device 1

[-]
[✓]
[X]

Connection Type	General MQTT
MQTT Broker	172.17.98.43
Tcp Port	1883
ClientID	
Cycle Time (ms)	1000
Watchdog Mode	Disabled
Watchdog Timeout (ms)	5000
Retain	Allow retained messages
Data Format	Binary
Main Topic	EK9160
Publish Topic	EK9160/EK-58CE72/Stream1/Bin/Tx/Data
Subscribe Topic	EK9160/EK-58CE72/Stream1/Bin/Rx/Data

Solution Explorer

Search Solution Explorer (Ctrl+ü)

Solution 'TwinCAT Project1' (1 project)

- ▲ TwinCAT Project1
 - ▶ SYSTEM
 - ▶ MOTION
 - ▶ PLC
 - ▶ SAFETY
 - ▶ C++
 - ▶ ANALYTICS
 - ▶ I/O
 - ▲ Devices
 - Device 1 (lot Mqtt Controller)
 - ▶ Image
 - ▶ Inputs
 - ▶ Outputs
 - ▶ Mappings

TwinCAT Project1

General

Mqtt

TLS

Topic

Main:

Device:

Stream:

Topic:

Broker

Port:

☒ Ip Address

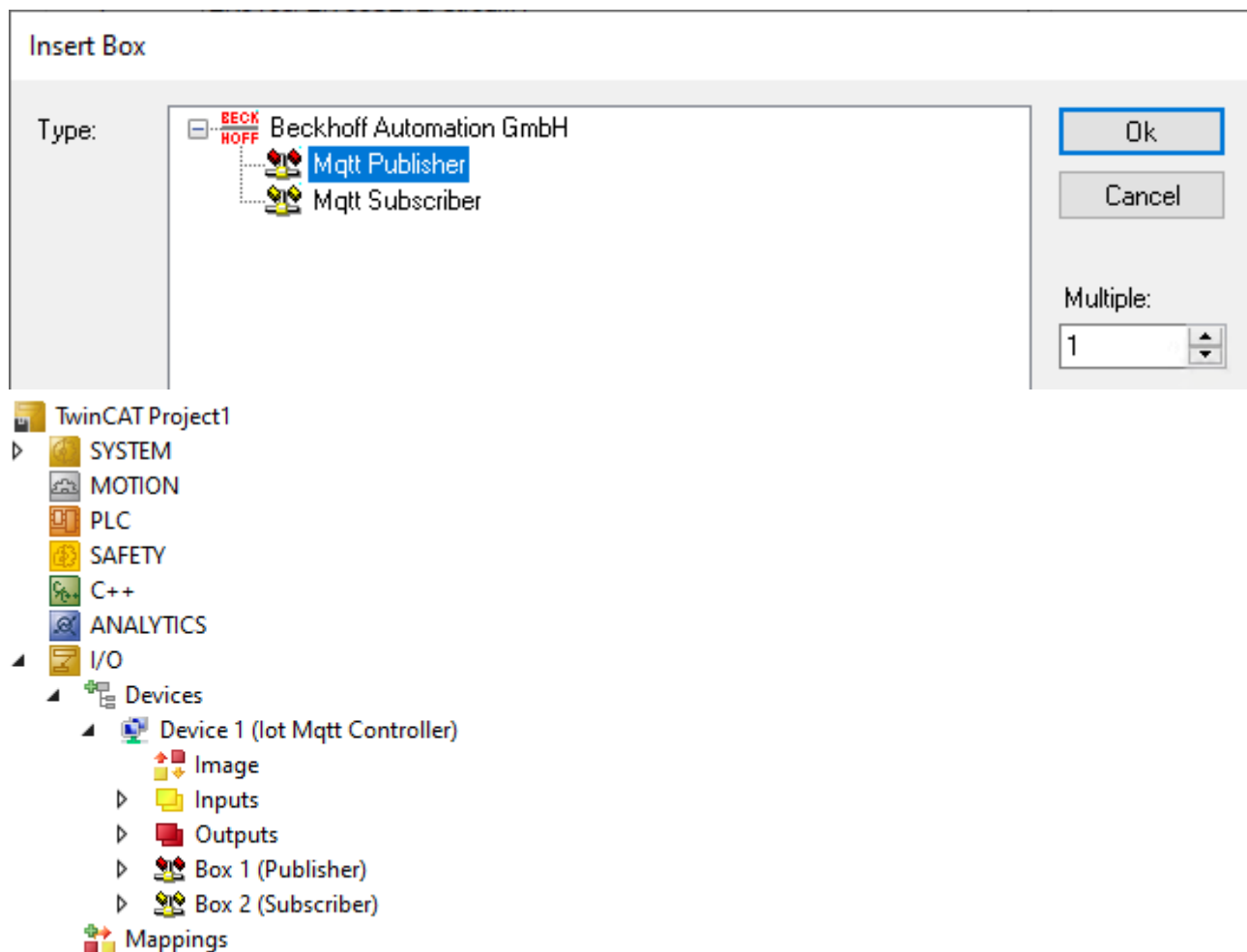
172 . 17 . 98 . 43

☐ Hostname

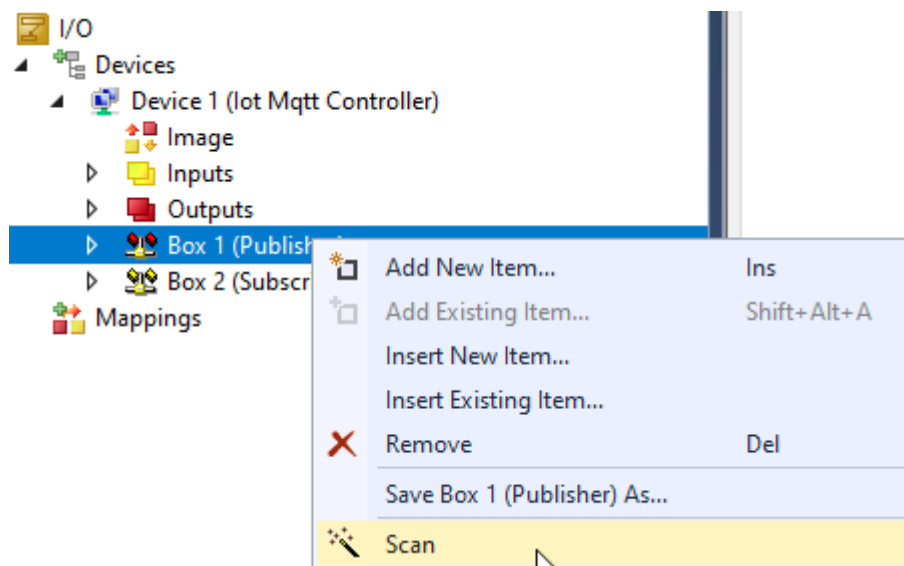
Username:

Password:

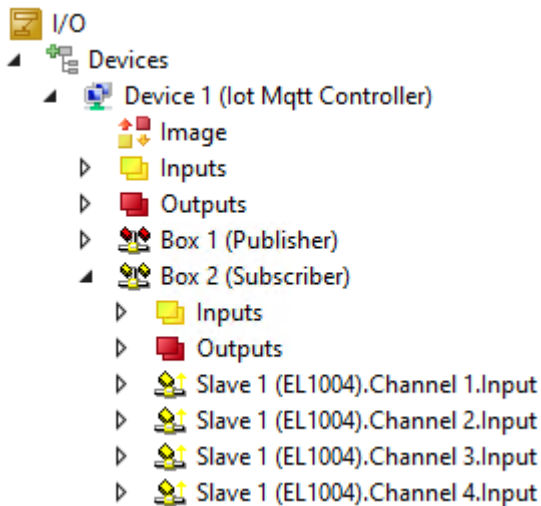
Publishers and Subscribers can then be created below the IoT MQTT controller, depending on whether you want to scan the output or input terminals. Input terminals are operated via the Publisher and output terminals via the Subscriber.



Subsequently, the symbol information can be read out via a scan mechanism and corresponding input/output variables can be automatically created in the process image of the device.



Result (taking the subscriber as an example):



Further Information

After the configuration has been activated on the EK9160, three topics below the configured Main Topic are used on the message broker:

1. The Symbol Topic contains the symbol information for the connected I/O terminals and is filled by the EK9160 after the communication link with the message broker has been established.
2. The Description Topic contains general status information about the device and is filled by the EK9160 after the communication connection with the message broker has been established.
3. The Data Topic contains the pure process data of the connected I/O terminals. This topic is thus cyclically filled with data by the EK9160.

The following screenshot shows a section of the Mosquitto Message Broker in verbose mode, on which you can see the individual publishes of the EK9160 on the above-mentioned topics.

```
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 LastWill handling

The LastWill is a message sent by the broker to all clients subscribed to the matching topic in case of an irregular connection failure. If the MQTT client in the PLC loses the connection to the broker and a LastWill was stored when the connection was established, this LastWill is communicated by the broker without the client having to worry about it.

In the case of a planned disconnect, the LastWill is not necessarily transmitted according to the specification. From the PLC programmer's point of view, he can decide whether he wants to publish the LastWill before calling the disconnect. For this purpose, the LastWill message is published again on the LastWill topic. This is necessary because the broker would not publish the LastWill message due to the regular disconnection.

In the event of a TwinCAT context change and a resulting restart of the MQTT communication, the IoT driver sends the previously specified LastWill to the broker, because at this moment there is no longer any possibility from the PLC. If no LastWill was defined when the connection was established, no message will be transmitted before the disconnect.

5 PLC API

5.1 Tc3_IotBase

5.1.1 FB_IotMqttClient

The function block enables communication with an MQTT broker.

A client function block is responsible for the connection to precisely one broker. The [Execute\(\)](#) [► 42] method of the function block must be called cyclically in order to ensure the background communication with this broker and facilitate receiving of messages.

All connection parameters exist as input parameters and are evaluated when a connection is established.

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 internally)
    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

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

Inputs

Name	Type	Description
sClientId	STRING(255)	The client ID can be specified individually and must be unique for most message brokers. If not specified, an ID based on the PLC project name is automatically generated by the TwinCAT driver.
sHostName	STRING(255)	sHostName can be specified as name or as IP address. If no information is provided, the local host is used.
nHostPort	UINT	The host port is specified here. The default is 1883.
sTopicPrefix	STRING(255)	Here you can specify a topic prefix that will be added automatically for all publish and subscribe commands.
nKeepAlive	UINT	Here you can specify the watchdog time (in seconds), with which the connection between client and broker is monitored.
sUserName	STRING(255)	Optionally, a user name can be specified.
sUserPassword	STRING(255)	A password for the user name can be entered here.
stWill	ST_lotMqttWill [► 46]	If the client is disconnected from the broker irregularly, a last preconfigured message can optionally be sent from the broker to the so-called "will topic". This message is specified here.
stTLS	ST_lotMqttTls [► 47]	If the broker offers a TLS-secured connection, the required configuration can be implemented here.
ipMessageFiFo	I_lotMqttMessageFiFo	<p>An instance of FB_lotMqttMessageQueue [► 48] can optionally be assigned here.</p> <p>This input parameter can be used for storing incoming new messages in the message queue without implementing the callback method. (See also lotMqttSampleUsingQueue [► 135])</p> <p>No message queue is used if the callback method is used or overwritten, irrespective of whether ipMessageQueue was set.</p>

Outputs

Name	Type	Description
bError	BOOL	Becomes TRUE as soon as an error situation occurs.
hrErrorCode	HRESULT	Returns an error code if the bError output is set. An explanation of the possible error codes can be found in the Appendix.
eConnectionState	ETclotMqttClientState	Indicates the state of the connection between client and broker as enumeration ETclotMqttClientState.
bConnected	BOOL	This output is TRUE if a connection exists between client and broker.

Methods

Name	Description
Execute [► 42]	Method for background communication with the TwinCAT driver. The method must be called cyclically.
Publish [► 43]	Method for executing a publish operation to a MQTT message broker.
Subscribe [► 44]	Method for establishing a subscription.
Unsubscribe [► 44]	Method for removing a subscription.
ActivateExponentialBackoff [► 46]	Activates the exponential backoff [► 27] function
DeactivateExponentialBackoff [► 46]	Deactivates the exponential backoff [► 27] function

Event-driven methods (callback methods)

Name	Description
OnMqttMessage [► 45]	Callback method used by the TwinCAT driver when a subscription to a topic was established and incoming messages are received.

Message payload formatting

Note that the data type and the formatting of the content must be known to the sender and receiver side, particularly when binary information (alignment) or strings (with or without zero termination) are sent.

Strings in UTF-8 format

The variables of type STRING used here are based on the UTF-8 format. This STRING formatting is common for MQTT communication as well as for JSON documents.

In order to be able to receive special characters and texts from a wide range of languages, the character set in the Tc3_lotBase and Tc3_JsonXml libraries is not limited to the typical character set of the data type STRING. Instead, the Unicode character set in UTF-8 format is used in conjunction with the data type STRING.

If the ASCII character set is used, there is no difference between the typical formatting of a STRING and the UTF-8 formatting of a STRING.

Further information on the UTF-8 STRING format and available display and conversion options can be found in the [documentation for the Tc2 Utilities PLC library](#).

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.0	IPC or CX (x86, x64, ARM)	Tc3_lotBase

5.1.1.1 Execute

This method must be called cyclically in order to ensure the background communication with the MQTT broker.

Syntax

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

Inputs

Name	Type	Description
bConnect	BOOL	The connection to the broker is established when bConnect is set to TRUE. bConnect must remain set to maintain the connection.

Any errors are reported at the outputs bError, hrErrorCode and eConnectionState of the function block instance.

5.1.1.2 Publish

This method is called once, in order to send a message to the broker.

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 subscribers
    bQueue      : BOOL; // for future extension
END_VAR
```

Inputs

Name	Type	Description
sTopic	STRING	Topic of the MQTT message
pPayload	PVOID	Address for the payload of the MQTT message
nPayloadSize	UDINT	Size of the payload in bytes
eQoS	TcIotMqttQos	"Quality of Service"
bRetain	BOOL	If bRetain is TRUE, the broker stores the message in order to make it available to subsequent subscribers.
bQueue	BOOL	Reserved parameter that can always be given the value FALSE.

Return value

Name	Type	Description
Publish	BOOL	The method returns the return value TRUE if the call was successful.

Possible errors are output at the outputs bError and hrErrorCode of the function block instance.

Message payload formatting

i Note that the data type and the formatting of the content must be known to the sender and receiver side, particularly when binary information (alignment) or strings (with or without zero termination) are sent.

Strings in UTF-8 format

The variables of type STRING used here are based on the UTF-8 format. This STRING formatting is common for MQTT communication as well as for JSON documents.

In order to be able to receive special characters and texts from a wide range of languages, the character set in the Tc3_IotBase and Tc3_JsonXml libraries is not limited to the typical character set of the data type STRING. Instead, the Unicode character set in UTF-8 format is used in conjunction with the data type STRING.

If the ASCII character set is used, there is no difference between the typical formatting of a STRING and the UTF-8 formatting of a STRING.

Further information on the UTF-8 STRING format and available display and conversion options can be found in the [documentation for the Tc2 Utilities PLC library](#).

5.1.1.3 Subscribe

This method is used by the client to signal to the broker that it wants to receive all MQTT message with a particular topic. The method can register an MQTT client instance on multiple topics.

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
```

Inputs

Name	Type	Description
sTopic	STRING	Topic of the MQTT message
eQoS	TcIotMqttQos	"Quality of Service"

Return value

Name	Type	Description
Subscribe	BOOL	The method returns the return value TRUE if the call was successful.

Possible errors are output at the outputs bError and hrErrorCode of the function block instance.

5.1.1.4 Unsubscribe

This method is used by the client to signal to the broker that no further messages with the specified topic should be transferred to it.

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
```

Inputs

Name	Type	Description
sTopic	STRING	Topic for which no further messages should be received.

Return value

Name	Type	Description
Unsubscribe	BOOL	The method returns the return value TRUE if the call was successful.

Possible errors are output at the outputs bError and hrErrorCode of the function block instance.

5.1.1.5 OnMqttMessage

Callback method

This method must not be called by the user. Instead, the function block FB_IotMqttClient can be used to derive information and overwrite this method. While the Execute() method is called, the responsible TwinCAT driver can call the OnMqttMessage() method in the event of new incoming messages. In the event of several incoming messages the callback method is called several times, once per message. This must be taken into account when the method is implemented.

The application of the callback method is explained further in the sample [IotMqttSampleUsingCallback](#) [► 137].

Syntax

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

Inputs

Name	Type	Description
Topic	STRING	Topic of the received MQTT message
payload	PVOID	Address for the payload of the received MQTT message
length	UDINT	Size of the payload in bytes
qos	TcIotMqttQos	"Quality of Service"
repeated	BOOL	If the user did not respond with S_OK to the last OnMqttMessage() method call, the message is issued again in the context of the next Execute() call, and the parameter repeated is set. This indicates that the message was issued more than once.

Return value

Name	Type	Description
OnMqttMessage	HRESULT	The return value of the method should be S_OK, if the message was accepted. If the message is to be issued again in the context of the next Execute() call, the return value can be assigned S_FALSE.

5.1.1.6 ActivateExponentialBackoff

A feature referred to as "exponential backoff" can be used to avoid burdening the message broker with unnecessary connection requests in case of a connection error. In the event of a TLS connection error involving the message broker, the reconnect rate is adjusted multiplicatively. This function can be activated using the [ActivateExponentialBackoff\(\)](#) [► 46] method. The parameters of the method specify the minimum and maximum time for the algorithm. The minimum time describes the initial delay value for the new connection attempt. The maximum time describes the highest delay value. The delay values are doubled until the maximum value is reached. Once a connection has been established, the backoff rate is reset to the original value. The [DeactivateExponentialBackoff\(\)](#) [► 46] method can be used to deactivate this function programmatically.

Syntax

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

Inputs

Name	Type	Description
tMqttBackoffMinTime	TIME	Describes the initial delay value for the new connection attempt.
tMqttBackoffMaxTime	TIME	Describes the largest delay value. Once this value has been reached, all new connection attempts are made at these intervals.

5.1.1.7 DeactivateExponentialBackoff

A feature referred to as "exponential backoff" can be used to avoid burdening the message broker with unnecessary connection requests in case of a connection error. In the event of a TLS connection error involving the message broker, the reconnect rate is adjusted multiplicatively. This function can be activated using the [ActivateExponentialBackoff\(\)](#) [► 46] method. The parameters of the method specify the minimum and maximum time for the algorithm. The minimum time describes the initial delay value for the new connection attempt. The maximum time describes the highest delay value. The delay values are doubled until the maximum value is reached. Once a connection has been established, the backoff rate is reset to the original value. The [DeactivateExponentialBackoff\(\)](#) [► 46] method can be used to deactivate this function programmatically.

Syntax

```
METHOD DeactivateExponentialBackoff
```

5.1.2 ST_IotMqttWill

The following information can be used to specify a message that is to be sent as the last message from the client to the broker in the event of an irregular disconnection between client and broker.

Syntax

Definition:

```
TYPE ST_IotMqttWill :
STRUCT
    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
END_STRUCT
```

```

    bRetain : BOOL; // if TRUE the broker stores the message in order to send it to new subscribers
END_STRUCT
END_TYPE

```

Parameter

Name	Type	Description
sTopic	STRING(255)	Message topic
pPayload	PVOID	Address for the payload of the message
nPayloadSize	UDINT	Size of the payload in bytes
eQoS	TcIotMqttQos	The “Quality of Service” parameter offers the following setting options: QoS level 0, QoS level 1, QoS level 2 (see QoS [► 26])
bRetain	BOOL	If bRetain is TRUE, the broker stores the message in order to make it available to subsequent subscribers.



Message payload formatting

Note that the data type and the formatting of the content must be known to the sender and receiver side, particularly when binary information (alignment) or strings (with or without zero termination) are sent.

5.1.3 ST_IotMqttTLS

TLS security setting for the MQTT client.

Either CA (certificate authority) or PSK (PreSharedKey) can be used.

Syntax

Definition:

```

TYPE ST_IotMqttTls :
STRUCT
    sCA
    string (PEM)
    sCAPath
    sCert
    string (PEM)
    sKeyFile
    sKeyPwd
    sCrl
    or as string (PEM)
    sCiphers
    sVersion
    bNoServerCertCheck : BOOL := FALSE;

    sPskIdentity
    aPskKey
    nPskKeyLen

    sAzureSas
END_STRUCT
END_TYPE

```

Parameter

Name	Type	Description
sCA	STRING(255)	Certificate of the certificate authority (CA)
sCert	STRING(255)	Client certificate to be used for authentication at the broker
sKeyFile	STRING(255)	Private key of the client
sKeyPwd	STRING(255)	Password of the private key, if applicable
sCrl	STRING(255)	Path to the certificate revocation list, which may be present in PEM or DER format
sCiphers	STRING(255)	Cipher suites to be used, specified in OpenSSL string format
sVersion	STRING(80)	TLS version to be used
bNoServerCertCheck	BOOL	Disables verification of the server certificate validity
sPskIdentity	STRING(255)	PreSharedKey identity for TLS PSK connection
aPskKey	ARRAY[1..64] OF BYTE	PreSharedKey for TLS PSK connection
nPskKeyLen	USINT	Length of the PreSharedKey in bytes
sAzureSAS	STRING(511)	SAS token for connection to the Microsoft Azure IoT Hub [► 15]

5.1.4 Message Queue

The use of this provided message queue is optional. Alternatively, users can implement a direct evaluation in the callback method.

5.1.4.1 FB_IotMqttMessageQueue

This function block offers a message queue for MQTT messages, which can be used with the block [FB_IotMqttClient](#) [► 40]. To this end an instance is declared and transferred at the input of [FB_IotMqttClient](#). The function block operates based on the first in, first out principle (FIFO).

During the program sequence it is possible to check whether messages were collected in the message queue, and how many. The `Dequeue()` method is used for removing messages from the FIFO queue. The oldest message is output first.

The number of MQTT messages currently held in the queue can be determined via the output `nQueuedMessages` (available as a property).

The input `bOverwriteOldestEntry` (also available as a property) can be used to specify whether a new messages should overwrite the oldest message when the queue is full. If yes (TRUE), the oldest message is lost. If no (FALSE), the latest message is lost. It is rare that the queue becomes full.

● Size of the MQTT message queue

1

The maximum number of possible messages in the queue can be set via the parameter `cMaxEntriesInMqttMessageQueue` in the parameter list of the library `Tc3_IotBase`. The default value is 1000 messages. This value can usually be left unchanged, since prompt message processing is required in most cases.

The MQTT message queue allocates new memory space for new messages according to the topic and payload size. By default the maximum size of a message is limited to 100 kB, the size of the MQTT message queue is limited to 1000 kB. For special cases these values can also be adjusted in the parameter list.

Methods

Name	Description
Dequeue() [▶ 49]	Removes an MQTT message from the queue
ResetQueue() [▶ 49]	Deletes all messages from the queue

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.0	IPC or CX (x86, x64, ARM)	Tc3_lotBase

5.1.4.1.1 Dequeue

The method returns the return value TRUE if the removal of an MQTT message from the queue was successful. The number of MQTT messages currently in the queue (nQueuedMessages property) is reduced by one through the removal of a message.

Syntax

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

Inputs

Name	Type	Description
fbMessage	REFERENCE TO FB_IotMqttMessage	For the message itself the reference is transferred to an instance of type FB_IotMqttMessage [▶ 49].

5.1.4.1.2 ResetQueue

When this method is called, all accumulated MQTT messages are deleted from the queue.

5.1.4.2 FB_IotMqttMessage

If the TwintCAT MQTT client (FB_IotMqttClient [[▶ 40](#)]) is used in combination with a message queue (FB_IotMqttMessageQueue [[▶ 48](#)]), an MQTT message is represented by the function block FB_IotMqttMessage. Incoming messages are collected by the message queue and made available to the user in the form of a such a function block instance.

The topic, the payload size and the "Quality of Service" parameter of the MQTT message are immediately available as Properties at the function block output. The topic and the payload can easily be evaluated or copied via the provided methods CompareTopic(), GetTopic() and GetPayload().

● Message payload formatting



Note that the data type and the formatting of the content must be known to the sender and receiver side, particularly when binary information (alignment) or strings (with or without zero termination) are sent.

● Size of an MQTT message



The maximum size of an MQTT message to be received in the PLC depends on the hardware platform and should not exceed a few MB, even on higher-performance/larger platforms.

The maximum message size can be set with the parameter `cMaxSizeOfMqttMessage` in the parameter list of the library `Tc3_lotBase`. By default, the message size is limited to 100 kB.

If the MQTT Message Queue is used, it dynamically allocates new memory space for new messages according to the topic and payload size.

🔧 Methods

Name	Description
<code>CompareTopic()</code> [► 50]	Compares a specified topic with the topic in the MQTT message
<code>GetTopic()</code> [► 51]	Returns the topic of an MQTT message
<code>GetPayload()</code> [► 51]	Returns the content of an MQTT message

● Strings in UTF-8 format



The variables of type `STRING` used here are based on the UTF-8 format. This `STRING` formatting is common for MQTT communication as well as for JSON documents.

In order to be able to receive special characters and texts from a wide range of languages, the character set in the `Tc3_lotBase` and `Tc3_JsonXml` libraries is not limited to the typical character set of the data type `STRING`. Instead, the Unicode character set in UTF-8 format is used in conjunction with the data type `STRING`.

If the ASCII character set is used, there is no difference between the typical formatting of a `STRING` and the UTF-8 formatting of a `STRING`.

Further information on the UTF-8 `STRING` format and available display and conversion options can be found in the [documentation for the Tc2 Utilities PLC library](#).

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.0	IPC or CX (x86, x64, ARM)	<code>Tc3_lotBase</code>

5.1.4.2.1 CompareTopic

This method returns `TRUE`, if the specified topic is identical to the topic of the MQTT message in the function block.

Syntax

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

🔧 Inputs

Name	Type	Description
<code>sTopic</code>	<code>STRING</code>	The topic to be compared is specified here.

5.1.4.2.2 GetTopic

Syntax

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

Inputs

Name	Type	Description
pTopic	POINTER TO STRING	The memory address for the buffer into which the topic is to be copied is specified here.
nTopicSize	UINT	The maximum available buffer size (in bytes) is specified here.

5.1.4.2.3 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
```

Inputs

Name	Type	Description
pPayload	PVOID	The memory address for the buffer into which the payload is to be copied is specified here.
nPayloadSize	UDINT	The maximum available buffer size (in bytes) is specified here.
bSetNullTermination	BOOL	If the payload type requires zero termination (string), this can be implemented during the copy process. This is not necessary if the message source (publisher) has already implemented a zero termination and this was taken into account in the payload size specification. In many cases no reliable information is available.

5.2 Tc3_JsonXml

5.2.1 Function blocks

5.2.1.1 FB_JsonDomParser

This function block is derived from the same internal function block as [FB_JsonDynDomParser \[► 81\]](#) and thus offers the same interface.

The two derived function blocks differ only in their internal memory management. FB_JsonDomParser is optimized for the fast and efficient parsing and creation of JSON documents that are only changed a little. The function block [FB_JsonDynDomParser \[► 81\]](#) is recommended for JSON documents to which many changes are made.

⚠ WARNING**Use of router memory**

The function block occupies new memory with each change, e.g. with the methods `SetObject()` or `SetJson()`. As a result, the amount of router memory used can grow enormously after repeated actions. This allocated memory is only released again by calling the method `NewDocument()` [► 701].

● Strings in UTF-8 format**1**

The variables of type `STRING` used here are based on the UTF-8 format. This `STRING` formatting is common for MQTT communication as well as for JSON documents.

In order to be able to receive special characters and texts from a wide range of languages, the character set in the `Tc3_lotBase` and `Tc3_JsonXml` libraries is not limited to the typical character set of the data type `STRING`. Instead, the Unicode character set in UTF-8 format is used in conjunction with the data type `STRING`.

If the ASCII character set is used, there is no difference between the typical formatting of a `STRING` and the UTF-8 formatting of a `STRING`.

Further information on the UTF-8 `STRING` format and available display and conversion options can be found in the [documentation for the Tc2 Utilities PLC library](#).

Requirements

TwinCAT version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4022	x86, x64, ARM	Tc3_JsonXml

5.2.1.1.1 AddArrayMember

This method adds an array member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.2 AddBase64Member

This method adds a Base64 member to a JSON object. A structure, for example, can be addressed as an input parameter. The corresponding Base64 coding is done by the method.

Syntax

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

Sample call:

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

5.2.1.1.3 AddBoolMember

This method adds a Bool member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.4 AddDateTimeMember

This method adds a DateTime member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.5 AddDcTimeMember

This method adds a DcTime member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.6 AddDoubleMember

This method adds a Double member to a JSON object.

Syntax

```

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

```

Sample call:

```

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

```

5.2.1.1.7 AddFileTimeMember

This method adds a FileTime member to a JSON object.

Syntax

```

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

```

Sample call:

```

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

```

5.2.1.1.8 AddHexBinaryMember

This method adds a HexBinary member to a JSON object.

Syntax

```

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

```

Sample call:

```

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

```

5.2.1.1.9 AddInt64Member

This method adds an Int64 member to a JSON object.

Syntax

```

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

```

Sample call:

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

5.2.1.1.10 AddIntMember

This method adds an Int member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.11 AddJsonMember

This method adds a JSON member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.12 AddNullMember

This method adds a NULL member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.13 AddObjectMember

This method adds an Object member to a JSON object.

Syntax

```

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

```

Sample call:

```

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

```

5.2.1.1.14 AddStringMember

This method adds a String member to a JSON object.

Syntax

```

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

```

Sample call:

```

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

```

5.2.1.1.15 AddUInt64Member

This method adds an UInt64 member to a JSON object.

Syntax

```

METHOD AddUInt64Member : SJsonValue
VAR_INPUT
    v : SJsonValue;
    value : ULINT;
END_VAR
VAR_IN_OUT CONSTANT
    member : STRING;
END_VAR

```

Sample call:

```

jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);
jsonMem := fbJson.AddUInt64Member(jsonDoc, 'TestUInt64', 42);

```

5.2.1.1.16 AddUIntMember

This method adds an UInt member to a JSON object.

Syntax

```

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

```

Sample call:

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

5.2.1.1.17 ArrayBegin

This method returns the first element of an array and can be used together with the methods `ArrayEnd()` and `NextArray()` for iteration through a JSON array.

Syntax

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

Sample call:

```
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

This method returns the last element of an array and can be used together with the methods `ArrayBegin()` and `NextArray()` for iteration through a JSON array.

Syntax

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

Sample call:

```
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

This method deletes the content of an array.

Syntax

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

Sample call:

The following JSON document is to be loaded into the DOM memory:

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

The values of the JSON array "array" are to be deleted. First of all, the JSON document is searched iteratively for the "array" property, after which all elements of the array are deleted by calling the `ClearArray()` method.

```

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

This method copies the contents of the DOM memory into a variable of data type STRING, which can have any length. The method returns the length of the string (including null termination). If the target buffer is too small, it is emptied by a null termination and returned as length 0.

Syntax

```

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

```

Sample call:

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

5.2.1.1.21 CopyJson

This method extracts a JSON object from a key and stores it in a variable of data type STRING. This STRING can be of any length. The method returns the length of the copied JSON object (including null termination). If the target buffer is too small, it is emptied by a null termination and returned as length 0.

Syntax

```

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

```

Sample call:

The following JSON document is to be loaded into the DOM memory:

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

The value of the JSON object "meta" is to be extracted and stored in a variable of data type STRING. First the JSON document is searched iteratively for the property "meta", then its value or sub-object is extracted by calling the method CopyJson().

```

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

```

After this run, the sString variable has the following content:

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

5.2.1.1.22 CopyString

This method copies the value of a key into a variable of the data type STRING, which can be of any length. The method returns the length of the copied string (including null termination). If the target buffer is too small, it is emptied by a null termination and returned as length 0.

Syntax

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

Sample call:

The following JSON document is to be loaded into the DOM memory:

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

The value of the key "clickType" is to be extracted and stored in a variable of data type STRING. First, the JSON document is iteratively searched for the property "clickType".

```
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
```

After this run, the sString variable has the following content:

```
SINGLE
```

5.2.1.1.23 FindMember

This method searches for a specific property in a JSON document and returns it. 0 is returned if no corresponding property is found.

Syntax

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

Sample call:

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

5.2.1.1.24 FindMemberPath

This method searches for a specific property in a JSON document and returns it. The property is specified according to its path in the document. 0 is returned if no corresponding property is found.

Syntax

```

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

```

Sample call:

```

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

```

5.2.1.1.25 GetArraySize

This method returns the number of elements in a JSON array.

Syntax

```

METHOD GetArraySize : UDINT
VAR_INPUT
    v : SJsonValue;
END_VAR

```

Sample call:

```

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

```

5.2.1.1.26 GetArrayValue

This method returns the value at the current iterator position of an array.

Syntax

```

METHOD GetArrayValue : SJsonValue
VAR_INPUT
    i : SJsonAIterator;
END_VAR

```

Sample call:

```

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

```

5.2.1.1.27 GetArrayValueByIdx

This method returns the value of an array in a specified index.

Syntax

```

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

```

Sample call:

```

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

```

5.2.1.1.28 GetBase64

This method decodes a Base64 value from a JSON property. If the content of a data structure, for example, is located behind the Base64 value, the decoded content can also be placed on an identical structure again.

Syntax

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

Sample call:

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

5.2.1.1.29 GetBool

This method returns the value of a property of the data type BOOL.

Syntax

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

5.2.1.1.30 GetDateTime

This method returns the value of a property of the data type DATE_AND_TIME.

Syntax

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

5.2.1.1.31 GetDcTime

This method returns the value of a property of the data type DCTIME.

Syntax

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

Sample call:

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

5.2.1.1.32 GetDocument

This method returns the content of the DOM memory as the data type STRING(255). With longer strings, the method will return a NULL string. In this case the method [CopyDocument](#) [► 58]() must be used.

Syntax

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

Sample call:

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

5.2.1.1.33 GetDocumentLength

This method returns the length of a JSON document in the DOM memory.

Syntax

```
METHOD GetDocumentLength: UDINT
```

Sample call:

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

5.2.1.1.34 GetDocumentRoot

This method returns the root node of a JSON document in the DOM memory.

Syntax

```
METHOD GetDocumentRoot : SJsonValue
```

Sample call:

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

5.2.1.1.35 GetDouble

This method returns the value of a property of the data type LREAL.

Syntax

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

5.2.1.1.36 GetFileTime

This method returns the value of a property of the data type DCTIME.

Syntax

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

Sample call:

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

5.2.1.1.37 GetHexBinary

This method decodes the HexBinary content of a property and writes it to a certain memory address, e.g. to a data structure.

Syntax

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

Sample call:

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

5.2.1.1.38 GetInt

This method returns the value of a property of the data type DINT.

Syntax

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

5.2.1.1.39 GetInt64

This method returns the value of a property of the data type LINT.

Syntax

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

5.2.1.1.40 GetJson

This method returns the value of a property as data type STRING(255), if this is a JSON document itself. With longer strings, the method will return a NULL string. In this case the method [CopyJson \[► 58\]\(\)](#) must be used.

Syntax

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

Sample call:

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

5.2.1.1.41 GetJsonLength

This method returns the length of a property if this is a JSON document.

Syntax

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

Sample call:

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

5.2.1.1.42 GetMaxDecimalPlaces

This method returns the current setting for MaxDecimalPlaces. This influences the number of decimal places in the case of floating point numbers.

Syntax

```
METHOD GetMaxDecimalPlaces : DINT
```

Sample call:

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

5.2.1.1.43 GetMemberName

This method returns the name of a JSON property member at the position of the current iterator, e.g. during the iteration of a child element of a JSON property with the methods MemberBegin(), MemberEnd() and NextMember().

Syntax

```
METHOD GetMemberName : STRING
VAR_INPUT
    i : SJsonIterator;
END_VAR
```

Sample call:

```
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
```

5.2.1.1.44 GetMemberValue

This method returns the value of a JSON property member at the position of the current iterator, e.g. during the iteration of a child element of a JSON property with the methods MemberBegin(), MemberEnd() and NextMember().

Syntax

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

Sample call:

```
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
```

5.2.1.1.45 GetString

This method returns the value of a property of the data type STRING(255). With longer strings, the method will return a NULL string. In this case the method [CopyString \[► 59\]\(\)](#) must be used.

Syntax

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

5.2.1.1.46 GetStringLength

This method returns the length of a property if its value is a string.

Syntax

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

Sample call:

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

5.2.1.1.47 GetType

This method returns the type of a property value. The return value can assume one of the values of the enum EJsonType.

Syntax

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

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

Sample call:

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

5.2.1.1.48 GetUint

This method returns the value of a property of the data type UDINT.

Syntax

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

5.2.1.1.49 GetUint64

This method returns the value of a property of the data type ULINT.

Syntax

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

5.2.1.1.50 HasMember

This method checks whether a certain property is present in the DOM memory. If the property is present the method returns TRUE, otherwise it returns FALSE.

Syntax

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

Sample call:

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

5.2.1.1.51 IsArray

This method checks whether a given property is an array. If that is the case, the method returns TRUE, otherwise it returns FALSE.

Syntax

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

5.2.1.1.52 IsBase64

This method checks whether the value of a given property is of the data type Base64. If that is the case, the method returns TRUE, otherwise it returns FALSE.

Syntax

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

5.2.1.1.53 IsBool

This method checks whether the value of a given property is of the data type BOOL. If that is the case, the method returns TRUE, otherwise it returns FALSE.

Syntax

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

5.2.1.1.54 IsDouble

This method checks whether the value of a given property is of the data type Double (PLC: LREAL). If that is the case, the method returns TRUE, otherwise it returns FALSE.

Syntax

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

5.2.1.1.55 IsFalse

This method checks whether the value of a given property is FALSE. If that is the case, the method returns TRUE, otherwise it returns FALSE.

Syntax

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

5.2.1.1.56 IsHexBinary

This method checks whether the value of a property is in the HexBinary format. If that is the case, the method returns TRUE, otherwise it returns FALSE.

Syntax

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

Sample call:

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

5.2.1.1.57 IsInt

This method checks whether the value of a given property is of the data type Integer (PLC: DINT). If that is the case, the method returns TRUE, otherwise it returns FALSE.

Syntax

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

5.2.1.1.58 IsInt64

This method checks whether the value of a given property is of the data type LINT. If that is the case, the method returns TRUE, otherwise it returns FALSE.

Syntax

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

5.2.1.1.59 IsISO8601TimeFormat

This method checks whether the value of a given property has a time format according to ISO8601. If that is the case, the method returns TRUE, otherwise it returns FALSE.

Syntax

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

5.2.1.1.60 IsNull

This method checks whether the value of a given property is NULL. If that is the case, the method returns TRUE, otherwise it returns FALSE.

Syntax

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

5.2.1.1.61 IsNumber

This method checks whether the value of a given property is a numerical value. If that is the case, the method returns TRUE, otherwise it returns FALSE.

Syntax

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

5.2.1.1.62 IsObject

This method checks whether the given property is a further JSON object. If that is the case, the method returns TRUE, otherwise it returns FALSE.

Syntax

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

5.2.1.1.63 IsString

This method checks whether the value of a given property is of the data type STRING. If that is the case, the method returns TRUE, otherwise it returns FALSE.

Syntax

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

5.2.1.1.64 IsTrue

This method checks whether the value of a given property is TRUE. If that is the case, the method returns TRUE, otherwise it returns FALSE.

Syntax

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

5.2.1.1.65 IsUInt

This method checks whether the value of a given property is of the data type UDINT. If that is the case, the method returns TRUE, otherwise it returns FALSE.

Syntax

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

5.2.1.1.66 IsUInt64

This method checks whether the value of a given property is of the data type ULINT. If that is the case, the method returns TRUE, otherwise it returns FALSE.

Syntax

```
METHOD IsUInt64 : BOOL
VAR_INPUT
    v : SJsonValue;
END_VAR
```

5.2.1.1.67 LoadDocumentFromFile

This method loads a JSON document from a file.

A rising edge on the input parameter bExec triggers the loading procedure. The asynchronous process is terminated as soon as the reference bExec is set back to FALSE from the method. When the process ends, the return value of the method indicates for one call whether the loading of the file was successful (TRUE) or failed (FALSE).

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
```

Sample call:

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

5.2.1.1.68 MemberBegin

This method returns the first child element below a JSON property and can be used by a JSON property together with the methods MemberEnd() and NextMember() for iteration.

Syntax

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

Sample call:

```
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
```

5.2.1.1.69 MemberEnd

This method returns the last child element below a JSON property and can be used by a JSON property together with the methods MemberBegin() and NextMember() for iteration.

Syntax

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

Sample call:

```
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
```

5.2.1.1.70 NewDocument

This method generates a new empty JSON document in the DOM memory.

Syntax

```
METHOD NewDocument : SJsonValue
```

Sample call:

```
jsonDoc := fbJson.NewDocument();
```

5.2.1.1.71 NextArray

5.2.1.1.72 ParseDocument

This method loads a JSON object into the DOM memory for further processing. The JSON object takes the form of a string and is transferred to the method as an input. A reference to the JSON document in the DOM memory is returned to the caller.

Syntax

```
METHOD ParseDocument : SJsonValue  
VAR_IN_OUT CONSTANT  
    sJson : STRING;  
END_VAR
```

Sample call:

```
jsonDoc := fbJson.ParseDocument(sJsonString);
```

5.2.1.1.73 PushbackBase64Value

This method appends a Base64 value to the end of an array. A structure, for example, can be addressed as an input parameter. The corresponding Base64 coding is done by the method.

Syntax

```
METHOD PushbackBase64Value : SJsonValue  
VAR_INPUT  
    v : SJsonValue;  
    p : PVOID;  
    n : DINT;  
END_VAR
```

Sample call:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);  
jsonArray := fbJson.FindMember(jsonDoc, 'array');  
jsonValue := fbJson.PushbackBase64Value(jsonArray, ADR(stStruct), SIZEOF(stStruct));
```

5.2.1.1.74 PushbackBoolValue

This method appends a value of the data type BOOL to the end of an array.

Syntax

```
METHOD PushbackBoolValue : SJsonValue  
VAR_INPUT  
    v : SJsonValue;  
    value : BOOL;  
END_VAR
```

Sample call:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);  
jsonArray := fbJson.FindMember(jsonDoc, 'array');  
jsonValue := fbJson.PushbackBoolValue(jsonArray, TRUE);
```

5.2.1.1.75 PushbackDateTimeValue

This method appends a value of the data type DATE_AND_TIME to the end of an array.

Syntax

```
METHOD PushbackDateTimeValue : SJsonValue  
VAR_INPUT  
    v : SJsonValue;  
    value : DATE_AND_TIME;  
END_VAR
```

Sample call:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);  
jsonArray := fbJson.FindMember(jsonDoc, 'array');  
jsonValue := fbJson.PushbackDateTimeValue(jsonArray, dtTime);
```

5.2.1.1.76 PushbackDcTimeValue

This method appends a value of the data type DCTIME to the end of an array.

Syntax

```
METHOD PushbackDcTimeValue : SJsonValue
VAR_INPUT
    v      : SJsonValue;
    value  : DCTIME;
END_VAR
```

Sample call:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);
jsonArray := fbJson.FindMember(jsonDoc, 'array');
jsonValue := fbJson.PushbackDcTimeValue(jsonArray, dcTime);
```

5.2.1.1.77 PushbackDoubleValue

This method appends a value of the data type Double to the end of an array.

Syntax

```
METHOD PushbackDoubleValue : SJsonValue
VAR_INPUT
    v      : SJsonValue;
    value  : LREAL;
END_VAR
```

Sample call:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);
jsonArray := fbJson.FindMember(jsonDoc, 'array');
jsonValue := fbJson.PushbackDoubleValue(jsonArray, 42.42);
```

5.2.1.1.78 PushbackFileTimeValue

This method appends a value of the data type FILETIME to the end of an array.

Syntax

```
METHOD PushbackFileTimeValue : SJsonValue
VAR_INPUT
    v      : SJsonValue;
    value  : FILETIME;
END_VAR
```

Sample call:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);
jsonArray := fbJson.FindMember(jsonDoc, 'array');
jsonValue := fbJson.PushbackFileTimeValue(jsonArray, fileTime);
```

5.2.1.1.79 PushbackHexBinaryValue

This method appends a HexBinary value to the end of an array. The coding in the HexBinary format is executed by the method. A data structure, for example, can be used as the source.

Syntax

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

Sample call:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);
jsonArray := fbJson.FindMember(jsonDoc, 'array');
jsonValue := fbJson.PushbackHexBinaryValue(jsonArray, ADR(stStruct), SIZEOF(stStruct));
```

5.2.1.1.80 PushbackInt64Value

This method appends a value of the data type Int64 to the end of an array.

Syntax

```
METHOD PushbackInt64Value : SJsonValue
VAR_INPUT
    v : SJsonValue;
    value : LINT;
END_VAR
```

Sample call:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);
jsonArray := fbJson.FindMember(jsonDoc, 'array');
jsonValue := fbJson.PushbackInt64Value(jsonArray, 42);
```

5.2.1.1.81 PushbackIntValue

This method appends a value of the data type INT to the end of an array.

Syntax

```
METHOD PushbackIntValue : SJsonValue
VAR_INPUT
    v : SJsonValue;
    value : DINT;
END_VAR
```

Sample call:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);
jsonArray := fbJson.FindMember(jsonDoc, 'array');
jsonValue := fbJson.PushbackIntValue(jsonArray, 42);
```

5.2.1.1.82 PushbackJsonValue

This method appends a JSON document to the end of an array.

Syntax

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

Sample call:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);
jsonArray := fbJson.FindMember(jsonDoc, 'array');
jsonValue := fbJson.PushbackJsonValue(jsonArray, sJson);
```

5.2.1.1.83 PushbackNullValue

This method appends a NULL value to the end of an array.

Syntax

```
METHOD PushbackNullValue : SJsonValue
VAR_INPUT
    v : SJsonValue;
END_VAR
```

Sample call:

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

5.2.1.1.84 PushbackStringValue

This method appends a value of the data type DCTIME to the end of an array.

Syntax

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

Sample call:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);
jsonArray := fbJson.FindMember(jsonDoc, 'array');
jsonValue := fbJson.PushbackStringValue(jsonArray, sString);
```

5.2.1.1.85 PushbackUInt64Value

This method appends a value of the data type UInt64 to the end of an array.

Syntax

```
METHOD PushbackUInt64Value : SJsonValue
VAR_INPUT
    v : SJsonValue;
    value : ULINT;
END_VAR
```

Sample call:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);
jsonArray := fbJson.FindMember(jsonDoc, 'array');
jsonValue := fbJson.PushbackUInt64Value(jsonArray, 42);
```

5.2.1.1.86 PushbackUIntValue

This method appends a value of the data type UInt to the end of an array.

Syntax

```
METHOD PushbackUIntValue : SJsonValue
VAR_INPUT
    v : SJsonValue;
    value : UDINT;
END_VAR
```

Sample call:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);
jsonArray := fbJson.FindMember(jsonDoc, 'array');
jsonValue := fbJson.PushbackUIntValue(jsonArray, 42);
```

5.2.1.1.87 RemoveAllMembers

This method removes all child elements from a given property.

Syntax

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

Sample call:

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

5.2.1.1.88 RemoveArray

This method deletes the value of the current array iterator.

Syntax

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

Sample call:

The following JSON document is to be loaded into the DOM memory:

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

The first array position is to be deleted. First of all, the JSON document is searched iteratively for the "array" property, after which the first element of the array is removed by calling the RemoveArray() method.

```
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
```

5.2.1.1.89 RemoveMember

This method deletes the property at the current iterator.

Syntax

```
METHOD RemoveMember : BOOL
VAR_INPUT
    v : SJsonValue;
    i : SJsonIterator;
    keepOrder : BOOL;
END_VAR
```

Sample call:

The following JSON document is to be loaded into the DOM memory:

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

The "array" property is to be deleted. First of all, the JSON document is searched for the "array" property, after which the property is removed.

```
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
```

5.2.1.1.90 RemoveMemberByName

This method removes a child element from a given property. The element is referenced by its name.

Syntax

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

Sample call:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);
jsonProp := fbJson.FindMember(jsonDoc, 'property');
jsonValue := fbJson.RemoveMemberByName(jsonProp, 'ChildName');
```

5.2.1.1.91 SaveDocumentToFile

This method saves a JSON document in a file.

A rising edge at the input parameter bExec triggers the saving procedure. The asynchronous process is terminated as soon as the reference bExec is set back to FALSE from the method. When the process ends, the return value of the method indicates for one call whether saving of the file was successful (TRUE) or failed (FALSE).

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
```

Sample call:

```
IF bSave THEN
    bSaved := fbJson.SaveDocumentToFile(sFile, bSave);
END_IF
```

5.2.1.1.92 SetArray

This method sets the value of a property to the type "Array". New values can now be added to the array with the Pushback methods.

Syntax

```
METHOD SetArray : SJsonValue  
VAR_INPUT  
    v : SJsonValue;  
END_VAR
```

Sample call:

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

5.2.1.1.93 SetBase64

This method sets the value of a property to a Base64-coded value. A data structure, for example, can be used as the source. Coding to the Base64 format takes place inside the method.

Syntax

```
METHOD SetBase64 : SJsonValue  
VAR_INPUT  
    v : SJsonValue;  
    p : PVOID;  
    n : DINT;  
END_VAR
```

Sample call:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);  
jsonProp := fbJson.FindMember(jsonDoc, 'property');  
jsonValue := fbJson.SetBase64(jsonProp, ADR(stStruct), SIZEOF(stStruct));
```

5.2.1.1.94 SetBool

This method sets the value of a property to a value of the data type BOOL.

Syntax

```
METHOD SetBool : SJsonValue  
VAR_INPUT  
    v : SJsonValue;  
    value : BOOL;  
END_VAR
```

Sample call:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);  
jsonProp := fbJson.FindMember(jsonDoc, 'property');  
jsonValue := fbJson.SetBool(jsonProp, TRUE);
```

5.2.1.1.95 SetDateTime

This method sets the value of a property to a value of the data type DATE_AND_TIME.

Syntax

```
METHOD SetDateTime : SJsonValue  
VAR_INPUT  
    v : SJsonValue;  
    value : DATE_AND_TIME;  
END_VAR
```

Sample call:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);  
jsonProp := fbJson.FindMember(jsonDoc, 'property');  
jsonValue := fbJson.SetDateTime(jsonProp, dtTime);
```

5.2.1.1.96 SetDcTime

This method sets the value of a property to a value of the data type DCTIME.

Syntax

```
METHOD SetDcTime : SJsonValue
VAR_INPUT
    v      : SJsonValue;
    value  : DCTIME;
END_VAR
```

Sample call:

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

5.2.1.1.97 SetDouble

This method sets the value of a property to a value of the data type Double.

Syntax

```
METHOD SetDouble : SJsonValue
VAR_INPUT
    v      : SJsonValue;
    value  : LREAL;
END_VAR
```

Sample call:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);
jsonProp := fbJson.FindMember(jsonDoc, 'property');
jsonValue := fbJson.SetDouble(jsonProp, 42.42);
```

5.2.1.1.98 SetFileTime

This method sets the value of a property to a value of the data type FILETIME.

Syntax

```
METHOD SetFileTime : SJsonValue
VAR_INPUT
    v      : SJsonValue;
    value  : FILETIME;
END_VAR
```

Sample call:

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

5.2.1.1.99 SetHexBinary

This method sets the value of a property to a HexBinary-coded value. A data structure, for example, can be used as the source. Coding to the HexBinary format takes place inside the method.

Syntax

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

Sample call:

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

5.2.1.1.100 SetInt

This method sets the value of a property to a value of the data type INT.

Syntax

```
METHOD SetInt : SJsonValue  
VAR_INPUT  
    v : SJsonValue;  
    value : DINT;  
END_VAR
```

Sample call:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);  
jsonProp := fbJson.FindMember(jsonDoc, 'property');  
jsonValue := fbJson.SetInt(jsonProp, 42);
```

5.2.1.1.101 SetInt64

This method sets the value of a property to a value of the data type Int64.

Syntax

```
METHOD SetInt64 : SJsonValue  
VAR_INPUT  
    v : SJsonValue;  
    value : LINT;  
END_VAR
```

Sample call:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);  
jsonProp := fbJson.FindMember(jsonDoc, 'property');  
jsonValue := fbJson.SetInt64(jsonProp, 42);
```

5.2.1.1.102 SetJson

This method inserts a further JSON document into the value of a property.

Syntax

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

Sample call:

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

5.2.1.1.103 SetMaxDecimalPlaces

This method sets the current setting for MaxDecimalPlaces. This sets the maximum number of decimal places to be used with floating point numbers.

Syntax

```
METHOD SetMaxDecimalPlaces
VAR_INPUT
    dp : DINT;
END_VAR
```

Sample call:

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

5.2.1.1.104 SetNull

This method sets the value of a property to the value NULL.

Syntax

```
METHOD SetNull : SJsonValue
VAR_INPUT
    v : SJsonValue;
END_VAR
```

Sample call:

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

5.2.1.1.105 SetObject

This method sets the value of a property to the type "Object". This enables the nesting of JSON documents.

Syntax

```
METHOD SetDouble : SJsonValue
VAR_INPUT
    v : SJsonValue;
    value : LREAL;
END_VAR
```

Sample call:

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

5.2.1.1.106 SetString

This method sets the value of a property to a value of the data type STRING.

Syntax

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

Sample call:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);
jsonProp := fbJson.FindMember(jsonDoc, 'property');
jsonValue := fbJson.SetString(jsonProp, 'Hello World');
```

5.2.1.1.107 SetUInt

This method sets the value of a property to a value of the data type UInt.

Syntax

```
METHOD SetUint : SJsonValue
VAR_INPUT
    v      : SJsonValue;
    value  : UDINT;
END_VAR
```

Sample call:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);
jsonProp := fbJson.FindMember(jsonDoc, 'property');
jsonValue := fbJson.SetUint(jsonProp, 42);
```

5.2.1.1.108 SetUint64

This method sets the value of a property to a value of the data type UInt64.

Syntax

```
METHOD SetUint64 : SJsonValue
VAR_INPUT
    v      : SJsonValue;
    value  : ULINT;
END_VAR
```

Sample call:

```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);
jsonProp := fbJson.FindMember(jsonDoc, 'property');
jsonValue := fbJson.SetUint64(jsonProp, 42);
```

5.2.1.2 FB_JsonDynDomParser

This function block is derived from the same internal function block as [FB_JsonDomParser](#) [► 51] and thus offers the same interface.

The two derived function blocks differ only in their internal memory management. **FB_JsonDynDomParser** is optimized for JSON documents to which many changes are made. It releases the allocated memory again after the execution of an action, e.g. for the methods `SetObject()` or `SetJson()`.

● Strings in UTF-8 format



The variables of type `STRING` used here are based on the UTF-8 format. This `STRING` formatting is common for MQTT communication as well as for JSON documents.

In order to be able to receive special characters and texts from a wide range of languages, the character set in the `Tc3_lotBase` and `Tc3_JsonXml` libraries is not limited to the typical character set of the data type `STRING`. Instead, the Unicode character set in UTF-8 format is used in conjunction with the data type `STRING`.

If the ASCII character set is used, there is no difference between the typical formatting of a `STRING` and the UTF-8 formatting of a `STRING`.

Further information on the UTF-8 `STRING` format and available display and conversion options can be found in the [documentation for the Tc2 Utilities PLC library](#).

Requirements

TwinCAT version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4024.7	x86, x64, ARM	Tc3_JsonXml 3.3.8.0

5.2.1.3 FB_JsonSaxReader

Requirements

TwinCAT version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4022	x86, x64, ARM	Tc3_JsonXml

5.2.1.3.1 DecodeBase64

This method converts a Base64-formatted string to binary data. If the conversion was successful the method returns TRUE, otherwise it returns 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
```

Sample call:

```
bSuccess := fbJson.DecodeBase64('SGVsbG8gVHdpbkNBVA==', ADR(byteArray), byteArraySize);
```

5.2.1.3.2 DecodeDateTime

This method enables the generation of a PLC variable of the type DATE_AND_TIME or DT from a standardized ISO8601 time format (e.g. YYYY-MM-DDThh:mm:ss). DT corresponds to the number of seconds starting from the date 1970-01-01. If the conversion was successful the method returns TRUE, otherwise it returns FALSE.

Syntax

```
METHOD DecodeDateTime : BOOL
VAR_IN_OUT CONSTANT
    sDT : STRING;
END_VAR
VAR_OUTPUT
    nDT : DATE_AND_TIME;
    hrErrorCode : HRESULT;
END_VAR
```

Sample call:

```
bSuccess := fbJson.DecodeDateTime('2017-08-09T06:54:00', nDT => dateTime);
```

5.2.1.3.3 DecodeDcTime

This method enables the generation of a PLC variable of the type DCTIME from a standardized ISO8601 time format (e.g. YYYY-MM-DDThh:mm:ss). DCTIME corresponds to the number of nanoseconds starting from the date 2000-01-01. If the conversion was successful the method returns TRUE, otherwise it returns FALSE.

Syntax

```
METHOD DecodeDcTime : BOOL
VAR_IN_OUT CONSTANT
    sDC : STRING;
END_VAR
VAR_OUTPUT
```

```
nDc : DCTIME;
hrErrorCode : HRESULT;
END_VAR
```

Sample call:

```
bSuccess := fbJson.DecodeDcTime('2017-08-09T06:54:00', nDc => dcTime);
```

5.2.1.3.4 DecodeFileTime

This method enables the generation of a PLC variable of the type FILETIME from a standardized ISO8601 time format (e.g. YYYY-MM-DDThh:mm:ss). FILETIME corresponds to the number of nanoseconds starting from the date 1601-01-01 – measured in 100 nanoseconds. If the conversion was successful the method returns TRUE, otherwise it returns FALSE.

Syntax

```
METHOD DecodeDateTime : BOOL
VAR_IN_OUT CONSTANT
  sFT : STRING;
END_VAR
VAR_OUTPUT
  nFT : FILETIME;
  hrErrorCode : HRESULT;
END_VAR
```

Sample call:

```
bSuccess := fbJson.DecodeFileTime('2017-08-09T06:54:00', nFT => fileTime);
```

5.2.1.3.5 DecodeHexBinary

This method converts a string containing hexadecimal values into binary data. If the conversion was successful the method returns TRUE, otherwise it returns 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
```

Sample call:

```
bSuccess := fbJson.DecodeHexBinary('ABCE93A', ADR(byteArray), byteArraySize);
```

5.2.1.3.6 GetLastParseResult

Syntax

```
METHOD GetLastParseResult : BOOL;
VAR_INPUT
  pOffset : POINTER TO LINT;
  pError : POINTER TO DINT;
END_VAR
```

5.2.1.3.7 IsBase64

This method checks whether the transferred string corresponds to the Base64 format. If that is the case, the method returns TRUE, otherwise it returns FALSE.

Syntax

```
METHOD IsBase64 : BOOL
VAR_IN_OUT CONSTANT
    sBase64 : STRING;
END_VAR
```

Sample call:

```
bIsBase64 := fbJson.IsBase64('SGVsbG8gVHdpbkNBVA==');
```

5.2.1.3.8 IsHexBinary

This method checks whether the transferred string consists of hexadecimal values. If that is the case, the method returns TRUE, otherwise it returns FALSE.

Syntax

```
METHOD IsHexBinary : BOOL
VAR_IN_OUT CONSTANT
    sHex : STRING;
END_VAR
```

Sample call:

```
bSuccess := fbJson.IsHexBinary('ABCEF93A');
```

5.2.1.3.9 IsISO8601TimeFormat

This method checks whether the transferred string corresponds to the standardized ISO8601 time format. If that is the case, the method returns TRUE, otherwise it returns FALSE.

Syntax

```
METHOD IsISO8601TimeFormat : BOOL
VAR_IN_OUT CONSTANT
    sDT : STRING;
END_VAR
```

Sample call:

```
bSuccess := fbJson.IsISO8601TimeFormat('2017-08-09T06:54:00');
```

5.2.1.3.10 Parse

This method starts the SAX reader parsing procedure. The JSON object to be parsed and a reference to a function block, which was derived from the interface ITcJsonSaxHandler, are transferred as input parameters. This function block is then used for the callback methods of the SAX reader.

Syntax

```
METHOD Parse : BOOL
VAR_IN_OUT CONSTANT
    sJson : STRING;
END_VAR
VAR_INPUT
    ipHdl : ITcJsonSaxHandler;
END_VAR
VAR_OUTPUT
    hrErrorCode : HRESULT;
END_VAR
```

5.2.1.3.11 ParseValues

This method starts the SAX reader parsing procedure. The JSON object to be parsed and a reference to a function block, which was derived from the interface `ITcJsonSaxValues`, are transferred as input parameters. This function block is then used for the callback methods of the SAX reader. What is special about this method is that exclusively values are taken into account in the callback methods, i.e. there are no `OnKey()` or `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
```

5.2.1.4 FB_JsonSaxWriter

● Strings in UTF-8 format



The variables of type `STRING` used here are based on the UTF-8 format. This `STRING` formatting is common for MQTT communication as well as for JSON documents.

In order to be able to receive special characters and texts from a wide range of languages, the character set in the `Tc3_lotBase` and `Tc3_JsonXml` libraries is not limited to the typical character set of the data type `STRING`. Instead, the Unicode character set in UTF-8 format is used in conjunction with the data type `STRING`.

If the ASCII character set is used, there is no difference between the typical formatting of a `STRING` and the UTF-8 formatting of a `STRING`.

Further information on the UTF-8 `STRING` format and available display and conversion options can be found in the [documentation for the Tc2 Utilities PLC library](#).

Requirements

TwinCAT version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4022	x86, x64, ARM	Tc3_JsonXml

5.2.1.4.1 AddBase64

This method adds a value of the data type `Base64` to a property. Usually, a corresponding property was created beforehand with the method [AddKey\(\)](#) [► 87].

Syntax

```
METHOD AddBase64
VAR_INPUT
    pBytes : Pointer TO BYTE;
    nBytes : DINT;
END_VAR
```

5.2.1.4.2 AddBool

This method adds a value of the data type `BOOL` to a property. Usually, a corresponding property was created beforehand with the method [AddKey\(\)](#) [► 87].

Syntax

```
METHOD AddBool
VAR_INPUT
    value : BOOL;
END_VAR
```

Sample call:

```
fbJson.AddKey('bSwitch');
fbJson.AddBool(TRUE);
```

5.2.1.4.3 AddDateTime

This method adds a value of the data type DATE_AND_TIME to a property. Usually, a corresponding property was created beforehand with the method [AddKey\(\)](#) [► 87].

Syntax

```
METHOD AddDateTime
VAR_INPUT
    value : DATE_AND_TIME;
END_VAR
```

Sample call:

```
fbJson.AddKey('Timestamp');
fbJson.AddDateTime(dtTime); // dtTime is of type DATE_AND_TIME
```

5.2.1.4.4 AddDcTime

This method adds a value of the data type DCTIME to a property. Usually, a corresponding property was created beforehand with the method [AddKey\(\)](#) [► 87].

Syntax

```
METHOD AddDcTime
VAR_INPUT
    value : DCTIME;
END_VAR
```

Sample call:

```
fbJson.AddKey('Timestamp');
fbJson.AddDcTime(dcTime); // dcTime is of type DCTIME
```

5.2.1.4.5 AddDint

This method adds a value of the data type DINT to a property. Usually, a corresponding property was created beforehand with the method [AddKey\(\)](#) [► 87].

Syntax

```
METHOD AddDint
VAR_INPUT
    value : DINT;
END_VAR
```

Sample call:

```
fbJson.AddKey('nNumber');
fbJson.AddDint(42);
```

5.2.1.4.6 AddFileTime

This method adds a value of the data type FILETIME to a property. Usually, a corresponding property was created beforehand with the method [AddKey\(\)](#) [► 87].

Syntax

```
METHOD AddFileTime
VAR_INPUT
    value : FILETIME;
END_VAR
```

Sample call:

```
fbJson.AddKey('Timestamp');
fbJson.AddFileTime(ftTime); // ftTime is of type FILETIME
```

5.2.1.4.7 AddHexBinary

This method adds a hex binary value to a property. Usually, a corresponding property was created beforehand with the method [AddKey\(\)](#) [► 87].

Syntax

```
METHOD AddHexBinary
VAR_INPUT
    pBytes : POINTER TO BYTE;
    nBytes : DINT;
END_VAR
```

Sample call:

```
fbJson.AddKey('HexBinary');
fbJson.AddHexBinary(ADR(byteHexBin), SIZEOF(byteHexBin));
```

5.2.1.4.8 AddKey

This method adds a new property key at the current position of the SAX writer. The value of the new property is usually set afterwards. This can be done using one of the following methods, for example: [AddBase64](#) [► 85], [AddBool](#) [► 85], [AddDateTime](#) [► 86], [AddDcTime](#) [► 86], [AddDint](#) [► 86], [AddFileTime](#) [► 86], [AddHexBinary](#) [► 87], [AddLint](#) [► 89], [AddLreal](#) [► 90], [AddNull](#) [► 90], [AddRawArray](#) [► 90], [AddRawObject](#) [► 90], [AddReal](#) [► 91], [AddString](#) [► 91], [AddUdint](#) [► 91], [AddUlint](#) [► 91].

Syntax

```
METHOD AddKey
VAR_IN_OUT CONSTANT
    key : STRING;
END_VAR
```

Sample call:

```
fbJson.AddKey('PropertyName');
```

5.2.1.4.9 AddKeyBool

This method creates a new property key and at the same time a value of the data type BOOL.

Syntax

```
METHOD AddKeyBool
VAR_IN_OUT CONSTANT
    key : STRING;
END_VAR
VAR_INPUT
    value : BOOL;
END_VAR
```

Sample call:

```
fbJson.AddKeyBool('bSwitch', TRUE);
```

5.2.1.4.10 AddKeyDateTime

This method creates a new property key and at the same time a value of the data type DATE_AND_TIME.

Syntax

```
METHOD AddKeyDateTime
VAR_IN_OUT CONSTANT
    key : STRING;
END_VAR
VAR_INPUT
    value : DATE_AND_TIME;
END_VAR
```

Sample call:

```
fbJson.AddKeyDateTime('Timestamp', dtTime);
```

5.2.1.4.11 AddKeyDcTime

This method creates a new property key and at the same time a value of the data type DCTIME.

Syntax

```
METHOD AddKeyDcTime
VAR_IN_OUT CONSTANT
    key : STRING;
END_VAR
VAR_INPUT
    value : DCTIME;
END_VAR
```

Sample call:

```
fbJson.AddKeyDcTime('Timestamp', dcTime);
```

5.2.1.4.12 AddKeyFileTime

This method creates a new property key and at the same time a value of the data type FILETIME.

Syntax

```
METHOD AddKeyFileTime
VAR_IN_OUT CONSTANT
    key : STRING;
END_VAR
VAR_INPUT
    value : FILETIME;
END_VAR
```

Sample call:

```
fbJson.AddKeyFileTime('Timestamp', ftTime);
```

5.2.1.4.13 AddKeyLreal

This method creates a new property key and at the same time a value of the data type LREAL.

Syntax

```
METHOD AddKeyLreal
VAR_IN_OUT CONSTANT
    key : STRING;
END_VAR
VAR_INPUT
    value : LREAL;
END_VAR
```

Sample call:

```
fbJson.AddKeyLreal('PropertyName', 42.42);
```

5.2.1.4.14 AddKeyNull

This method creates a new property key and initializes its value with zero.

Syntax

```
METHOD AddKeyNull
VAR_IN_OUT CONSTANT
    key : STRING;
END_VAR
```

Sample call:

```
fbJson.AddKeyNull('PropertyName');
```

5.2.1.4.15 AddKeyNumber

This method creates a new property key and at the same time a value of the data type DINT.

Syntax

```
METHOD AddKeyNumber
VAR_IN_OUT CONSTANT
    key : STRING;
END_VAR
VAR_INPUT
    value : DINT;
END_VAR
```

Sample call:

```
fbJson.AddKeyNumber('PropertyName', 42);
```

5.2.1.4.16 AddKeyString

This method creates a new property key and at the same time a value of the data type STRING.

Syntax

```
METHOD AddKeyString
VAR_IN_OUT CONSTANT
    key : STRING;
    value : STRING;
END_VAR
```

Sample call:

```
fbJson.AddKeyString('PropertyName', 'Hello World');
```

5.2.1.4.17 AddLint

This method adds a value of the data type LINT to a property. Usually, a corresponding property was created beforehand with the method [AddKey\(\)](#) [► 87].

Syntax

```
METHOD AddLint
VAR_INPUT
    value : LINT;
END_VAR
```

Sample call:

```
fbJson.AddKey('PropertyName');
fbJson.AddLint(42);
```

5.2.1.4.18 AddLreal

This method adds a value of the data type LREAL to a property. Usually, a corresponding property was created beforehand with the method [AddKey\(\)](#) [► 87].

Syntax

```
METHOD AddLreal
VAR_INPUT
    value : LREAL;
END_VAR
```

Sample call:

```
fbJson.AddKey('PropertyName');
fbJson.AddLreal(42.42);
```

5.2.1.4.19 AddNull

This method adds the value zero to a property. Usually, a corresponding property was created beforehand with the method [AddKey\(\)](#) [► 87].

Syntax

```
METHOD AddNull
```

Sample call:

```
fbJson.AddKey('PropertyName');
fbJson.AddNull();
```

5.2.1.4.20 AddRawArray

This method adds a valid JSON array to a given property as a value. The array to be added must be in a valid JSON format and may only be added if the SAX writer is at a correspondingly valid position, i.e. for example, directly after a preceding [AddKey\(\)](#) [► 87], [StartArray\(\)](#) [► 93] or as the first call after a [ResetDocument\(\)](#) [► 93].

Syntax

```
METHOD AddRawArray
VAR_IN_OUT CONSTANT
    rawJson : STRING;
END_VAR
```

Sample call:

```
fbJson.AddKey('PropertyName');
fbJson.AddRawArray('[1, 2, {"x":42, "y":42}, 4]');
```

5.2.1.4.21 AddRawObject

This method adds a valid JSON object to a given property as a value. The object to be added must be in a valid JSON format and may only be added if the SAX writer is at a correspondingly valid position, i.e. for example, directly after a preceding [AddKey\(\)](#) [► 87], [StartArray\(\)](#) [► 93] or as the first call after a [ResetDocument\(\)](#) [► 93].

Syntax

```
METHOD AddRawObject
VAR_IN_OUT CONSTANT
    rawJson : STRING;
END_VAR
```

Sample call:

```
fbJson.AddKey('PropertyName');  
fbJson.AddRawObject({'x':42, 'y':42});
```

5.2.1.4.22 AddReal

This method adds a value of the data type REAL to a property. Usually, a corresponding property was created beforehand with the method [AddKey\(\)](#) [► 87].

Syntax

```
METHOD AddReal  
VAR_INPUT  
    value : REAL;  
END_VAR
```

Sample call:

```
fbJson.AddKey('PropertyName');  
fbJson.AddReal(42.42);
```

5.2.1.4.23 AddString

This method adds a value of the data type STRING to a property. Usually, a corresponding property was created beforehand with the method [AddKey\(\)](#) [► 87].

Syntax

```
METHOD AddString  
VAR_IN_OUT CONSTANT  
    value : STRING;  
END_VAR
```

Sample call:

```
fbJson.AddKey('PropertyName');  
fbJson.AddString('Hello World');
```

5.2.1.4.24 AddUdint

This method adds a value of the data type UDINT to a property. Usually, a corresponding property was created beforehand with the method [AddKey\(\)](#) [► 87].

Syntax

```
METHOD AddUdint  
VAR_INPUT  
    value : UDINT;  
END_VAR
```

Sample call:

```
fbJson.AddKey('PropertyName');  
fbJson.AddUdint(42);
```

5.2.1.4.25 AddUlint

This method adds a value of the data type ULINT to a property. Usually, a corresponding property was created beforehand with the method [AddKey\(\)](#) [► 87].

Syntax

```
METHOD AddUlint  
VAR_INPUT  
    value : ULINT;  
END_VAR
```

Sample call:

```
fbJson.AddKey('PropertyName');  
fbJson.AddUlint(42);
```

5.2.1.4.26 CopyDocument

This method copies the content of the current JSON object created with the SAX Writer to a target variable of the data type STRING, which can be of any length. The method returns the length of the string (including null termination). If the target buffer is too small, it is emptied by a null termination and returned as length 0.

Syntax

```
METHOD CopyDocument : UDINT  
VAR_IN_OUT CONSTANT  
    pDoc : STRING;  
END_VAR  
VAR_INPUT  
    nDoc : UDINT;  
END_VAR  
VAR_OUTPUT  
    hrErrorCode: HRESULT;  
END_VAR
```

Sample call:

```
fbJson.CopyDocument(sTargetString, SIZEOF(sTargetString));
```

5.2.1.4.27 EndArray

This method generates the end of a started JSON array ("square closing bracket") and inserts it at the current position of the SAX writer.

Syntax

```
METHOD EndArray : HRESULT
```

Sample call:

```
fbJson.EndArray();
```

5.2.1.4.28 EndObject

This method generates the end of a started JSON object ("curly closing bracket") and inserts it at the current position of the SAX writer.

Syntax

```
METHOD EndObject : HRESULT
```

Sample call:

```
fbJson.EndObject();
```

5.2.1.4.29 GetDocument

This method returns the content of the JSON object that is currently created with the SAX Writer and returns it as data type STRING(255).

The maximum size of the string returned by the method is 255 characters. With longer strings, the method will return a NULL string. In this case the method [CopyDocument](#) [► 92]() must be used.

Syntax

```
METHOD GetDocument : STRING(255)
VAR_OUTPUT
    hrErrorCode: HRESULT;
END_VAR
```

Sample call:

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

5.2.1.4.30 GetDocumentLength

This method returns the length of the JSON object that is currently created with the SAX Writer and returns it as data type UDINT.

Syntax

```
METHOD GetDocumentLength : UDINT
VAR_OUTPUT
    hrErrorCode: HRESULT;
END_VAR
```

Sample call:

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

5.2.1.4.31 GetMaxDecimalPlaces**Syntax**

```
METHOD GetMaxDecimalPlaces : DINT
```

5.2.1.4.32 ResetDocument

This method resets the JSON object currently created with the SAX writer.

Syntax

```
METHOD ResetDocument : HRESULT
```

Sample call:

```
fbJson.ResetDocument();
```

5.2.1.4.33 SetMaxDecimalPlaces**Syntax**

```
METHOD SetMaxDecimalPlaces : HRESULT
VAR_INPUT
    decimalPlaces: DINT;
END_VAR
```

5.2.1.4.34 StartArray

This method generates the start of a new JSON array ("square opening bracket") and inserts it at the current position of the SAX writer.

Syntax

```
METHOD StartArray : HRESULT
```

Sample call:

```
fbJson.StartArray();
```

5.2.1.4.35 StartObject

This method generates the start of a new JSON object ("curly opening bracket") and inserts it at the current position of the SAX writer.

Syntax

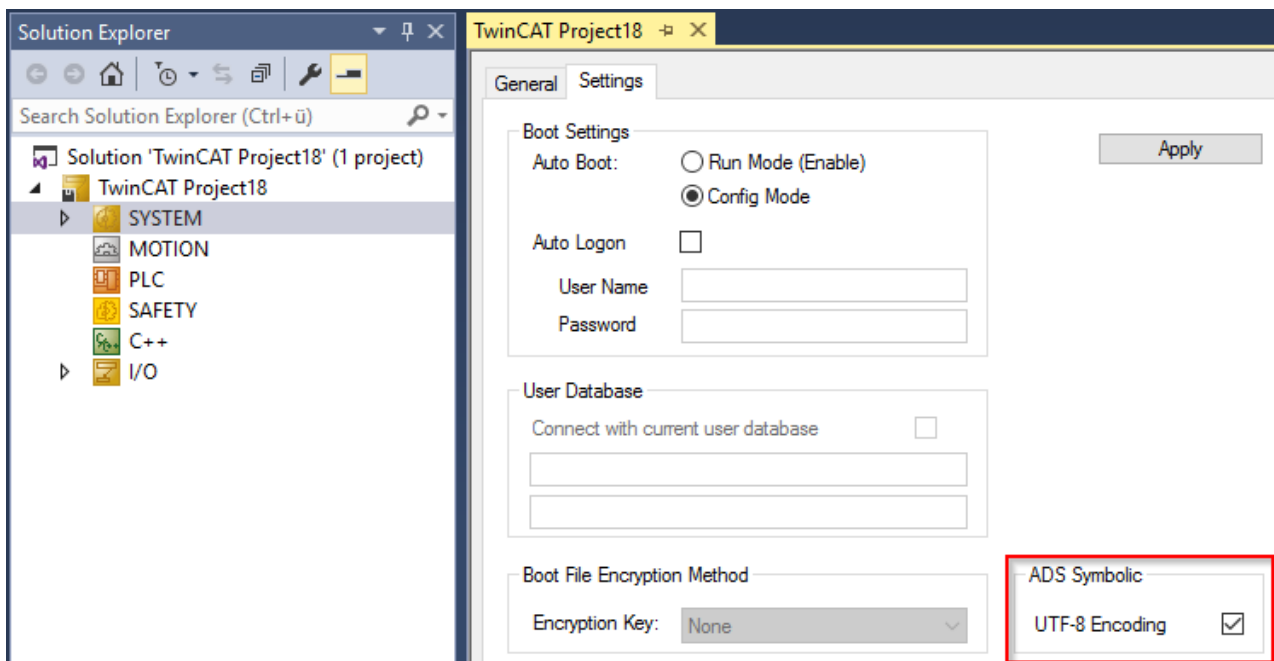
```
METHOD StartObject : HRESULT
```

Sample call:

```
fbJson.StartObject();
```

5.2.1.5 FB_JsonReadWriteDataType

In order to use UTF-8 characters, e.g. in the automatic generation of metadata via the function block [FB_JsonReadWriteDataType](#) [► 94], the check box for the support of UTF-8 in the symbolism must be activated in the TwinCAT project. To do this, double-click on **SYSTEM** in the project tree, open the **Settings** tab and activate the corresponding check box.



● Strings in UTF-8 format

The variables of type STRING used here are based on the UTF-8 format. This STRING formatting is common for MQTT communication as well as for JSON documents.

In order to be able to receive special characters and texts from a wide range of languages, the character set in the Tc3_lotBase and Tc3_JsonXml libraries is not limited to the typical character set of the data type STRING. Instead, the Unicode character set in UTF-8 format is used in conjunction with the data type STRING.

If the ASCII character set is used, there is no difference between the typical formatting of a STRING and the UTF-8 formatting of a STRING.

Further information on the UTF-8 STRING format and available display and conversion options can be found in the [documentation for the Tc2 Utilities PLC library](#).

Requirements

TwinCAT version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4022	x86, x64, ARM	Tc3_JsonXml

5.2.1.5.1 AddJsonKeyPropertiesFromSymbol

With the aid of this method, metadata can be added via PLC attributes to the JSON representation of a PLC data structure on an [FB_JsonSaxWriter](#) [► 85] object. The method receives as its input parameters the instance of the FB_JsonSaxWriter function block, the desired name of the JSON property that is to contain the metadata, the data type name of the structure and a string variable sProperties, which contains a list of the PLC attributes to be extracted, separated by a cross bar.

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
```

The PLC attributes can be specified in the following form on the structure elements:

```
{attribute 'Unit' := 'm/s'}
{attribute 'DisplayName' := 'Speed'}
Sensor1 : REAL;
```

A complete sample of how to use this method can be found in section [Tc3JsonXmlSampleJsonDataType](#) [► 153].

Sample call:

```
fbJsonSaxWriter.ResetDocument()
fbJsonDataType.AddJsonKeyPropertiesFromSymbol(fbJsonSaxWriter, 'MetaData', 'ST_Values', 'Unit|
DisplayName');
```

5.2.1.5.2 AddJsonKeyValueFromSymbol

This method generates the JSON representation of a PLC data structure on an [FB_JsonSaxWriter](#) [► 85] object. The method receives as its input parameters the instance of the FB_JsonSaxWriter function block, the data type name of the structure, and the address and size of the source structure instance. As a result, the FB_JsonSaxWriter instance contains a valid JSON representation of the structure. Unlike the method [AddJsonValueFromSymbol\(\)](#) [► 96], the elements of the source structure are nested here in a JSON sub-object whose name can be specified via the input/output parameter sKey.

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
```

A complete sample of how to use this method can be found in section [Tc3JsonXmlSampleJsonDataType](#) [► 153].

Sample call:

```
fbJsonSaxWriter.ResetDocument()
fbJsonDataType.AddJsonKeyValueFromSymbol(fbJsonSaxWriter, 'Values','ST_Values',SIZEOF(stValues),
ADR(stValues));
```

5.2.1.5.3 AddJsonValueFromSymbol

This method generates the JSON representation of a PLC data structure on an [FB JsonSaxWriter](#) [► 85] object. The method receives as its input parameters the instance of the FB JsonSaxWriter function block, the data type name of the structure, and the address and size of the source structure instance. As a result, the FB JsonSaxWriter instance contains a valid JSON representation of the structure.

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
```

A complete sample of how to use this method can be found in section [Tc3JsonXmlSampleJsonDataType](#) [► 153].

Sample call:

```
fbJsonSaxWriter.ResetDocument()
fbJsonDataType.AddJsonValueFromSymbol(fbJsonSaxWriter,'ST_Values',SIZEOF(stValues), ADR(stValues));
```

5.2.1.5.4 CopyJsonStringFromSymbol

This method generates the JSON representation of a symbol and copies it into a variable of the data type STRING, which can be of any length. The method returns the length of the string (including null termination). If the target buffer is too small, it is emptied by a null termination and returned as length 0.

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
```

Sample call:

```
nLen :=
fbJsonDataType.CopyJsonStringFromSymbol('ST_Test',SIZEOF(stTest),ADR(stTest),sString,SIZEOF(sString)
);
```

5.2.1.5.5 CopyJsonStringFromSymbolProperties

This method generates a corresponding JSON representation of PLC attributes on a symbol. In contrast to the [AddJsonKeyPropertiesFromSymbol](#) [► 95] method, the result is not written to an instance of the function block FB_JsonSaxWriter, but to a string variable. The method receives as its input parameters the data type name of the symbol and a string variable that represents a list of the PLC attributes to be extracted, separated by a cross bar.

The method copies this JSON representation into a variable of the data type STRING, which can be of any length. The method returns the length of the string (including null termination). If the target buffer is too small, it is emptied by a null termination and returned as length 0.

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
```

Sample call:

```
nLen := fbJsonDataType.CopyJsonStringFromSymbolProperties('ST_Test', 'Unit|
DisplayName', sString, SIZEOF(sString));
```

5.2.1.5.6 CopySymbolNameByAddress

This method returns the complete (ADS) symbol name of a transferred symbol. The method returns the size of the string (including null termination). If the target buffer is too small, it is emptied by a null termination and returned as length 0.

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
```

Sample call:

```
nSymbolSize := fbJsonDataType.CopySymbolNameByAddress(nData:=SIZEOF(stValues), pData:=ADR(stValues),
sName:=sSymbolName, nName:=SIZEOF(sSymbolName));
```

Requirements

TwinCAT version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4024.20	x86, x64, ARM	Tc3_JsonXml 3.3.15.0

5.2.1.5.7 GetDataTypeNameByAddress

This method returns the data type name of a transferred symbol.

Syntax

```

METHOD GetDataTypeNameByAddress : STRING
VAR_INPUT
    nData : UDINT;
    pData : PVOID;
END_VAR
VAR_OUTPUT
    hrErrorCode : HRESULT;
END_VAR

```

Sample call:

```
sBuffer := fbJsonDataType.GetDataTypeNameByAddress(SIZEOF(stValues), ADR(stValues));
```

5.2.1.5.8 GetJsonFromSymbol

This method generates the corresponding JSON representation of a symbol. In contrast to the [AddJsonValueFromSymbol\(\) \[► 96\]](#) method, the result is not written to an instance of the function block `FB_JsonSaxWriter`, but to a string variable. The method receives as its input parameters the data type name of the symbol as well as the address and size of the source symbol, e.g. of a structure instance. The address and size of the destination buffer that contains the JSON representation of the symbol after the call are transferred as further input parameters.

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

```

**Input parameter nJson**

The input parameter `nJson` contains the size of the target buffer when the method is called, and the size of the actually written JSON object in the target buffer when the method call is completed.

Sample call:

```
fbJsonDataType.GetJsonFromSymbol('ST_Values', SIZEOF(stValues), ADR(stValues), nBufferLength, ADR(sBuffer));
```

5.2.1.5.9 GetJsonStringFromSymbol

This method generates the corresponding JSON representation of a symbol. In contrast to the [AddJsonValueFromSymbol\(\) \[► 96\]](#) method, the result is not written to an instance of the function block `FB_JsonSaxWriter`, but to a string variable. The method receives as its input parameters the data type name of the symbol as well as the address and size of the source symbol, e.g., of a structure instance.

The maximum size of the string returned by the method is 255 characters. With longer strings, the method will return a NULL string. In this case the method [CopyJsonStringFromSymbol \[► 96\]\(\)](#) must be used.

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
```

Sample call:

```
sBuffer := fbJsonDataType.GetJsonStringFromSymbol('ST_Values', SIZEOF(stValues), ADR(stValues));
```

5.2.1.5.10 GetJsonStringFromSymbolProperties

This method generates a corresponding JSON representation of PLC attributes on a symbol. In contrast to the [AddJsonKeyPropertiesFromSymbol \[► 95\]](#) method, the result is not written to an instance of the function block FB_JsonSaxWriter, but to a string variable. The method receives as its input parameters the data type name of the symbol and a string variable that represents a list of the PLC attributes to be extracted, separated by a cross bar. The result is returned directly as the return value of the method.

The maximum size of the string returned by the method is 255 characters. With longer strings, the method will return a NULL string. In this case the method [CopyJsonStringFromSymbolProperties \[► 97\]\(\)](#) must be used.

Syntax

```
METHOD GetJsonStringFromSymbolProperties : STRING(255)
VAR_IN_OUT CONSTANT
    sDatatype : STRING;
    sProperties : STRING;
END_VAR
VAR_OUTPUT
    hrErrorCode : HRESULT;
END_VAR
```

Sample call:

```
sBuffer := fbJsonDataType.GetJsonStringFromSymbolProperties('ST_Values', 'Unit|DisplayName');
```

5.2.1.5.11 GetSizeJsonStringFromSymbol

This method reads the size of the JSON representation of a symbol. The value is specified with null termination.

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
```

Sample call:

```
nLen := fbJsonDataType.GetSizeJsonStringFromSymbol('BOOL', SIZEOF(bBool), ADR(bBool));
```

5.2.1.5.12 GetSizeJsonStringFromSymbolProperties

This method reads the size of the JSON representation of PLC attributes on a symbol. The value is specified with null termination.

Syntax

```
METHOD GetSizeJsonStringFromSymbolProperties : UDINT
VAR_IN_OUT CONSTANT
    sDatatype : STRING;
```

```

    sProperties : STRING;
END_VAR
VAR_OUTPUT
    hrErrorCode : HRESULT;
END_VAR

```

Sample call:

```
nLen := fbJsonDataType.GetSizeJsonStringFromSymbolProperties('ST_Test','DisplayName|Unit');
```

5.2.1.5.13 GetSymbolNameByAddress

This method returns the complete (ADS) symbol name of a transferred symbol.

The maximum size of the string returned by the method is 255 characters. With longer strings, the method will return a null string. In this case the method [CopySymbolNameByAddress\(\)](#) [► 97] must be used.

Syntax

```

METHOD GetSymbolNameByAddress : STRING(255)
VAR_INPUT
    nData : UDINT;
    pData : PVOID;
END_VAR
VAR_OUTPUT
    hrErrorCode : HRESULT;
END_VAR

```

Sample call:

```
sBuffer := fbJsonDataType.GetSymbolNameByAddress(SIZEOF(stValues), ADR(stValues));
```

5.2.1.5.14 SetSymbolFromJson

This method extracts a string containing a valid JSON message and attempts to save the contents of the JSON object to an equivalent data structure. The method receives as its input parameters the string with the JSON object, the data type name of the target structure, and the address and size of the target structure instance.

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

```

Sample call:

```
fbJsonDataType.SetSymbolFromJson(sJson,'ST_Values',SIZEOF(stValuesReceive), ADR(stValuesReceive));
```

5.2.1.6 FB_XmlDomParser

● Strings in UTF-8 format



The variables of type STRING used here are based on the UTF-8 format. This STRING formatting is common for MQTT communication as well as for JSON documents.

In order to be able to receive special characters and texts from a wide range of languages, the character set in the Tc3_lotBase and Tc3_JsonXml libraries is not limited to the typical character set of the data type STRING. Instead, the Unicode character set in UTF-8 format is used in conjunction with the data type STRING.

If the ASCII character set is used, there is no difference between the typical formatting of a STRING and the UTF-8 formatting of a STRING.

Further information on the UTF-8 STRING format and available display and conversion options can be found in the [documentation for the Tc2 Utilities PLC library](#).

Requirements

TwinCAT version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4022	x86, x64, ARM	Tc3_JsonXml

5.2.1.6.1 AppendAttribute

This method adds a new attribute to an existing node. The name and value of the new attribute and the existing XML node are transferred to the method as input parameters. The method returns a reference to the newly added attribute.

Syntax

```
METHOD AppendAttribute : SXmlAttribute
VAR_INPUT
    n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
    name : STRING;
    value : STRING;
END_VAR
```

Sample call:

```
objAttribute := fbXml.AppendAttribute(objMachine, 'Name', 'some value');
```

5.2.1.6.2 AppendAttributeAsBool

This method adds a new attribute to an existing node. The value of the attribute has the data type Boolean. The name and value of the new attribute and the existing XML node are transferred to the method as input parameters. The method returns a reference to the newly added attribute.

Syntax

```
METHOD AppendAttributeAsBool : SXmlAttribute
VAR_INPUT
    n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
    name : STRING;
END_VAR
VAR_INPUT
    value : BOOL;
END_VAR
```

Sample call:

```
objAttribute := fbXml.AppendAttributeAsBool(objMachine, 'Name', TRUE);
```

5.2.1.6.3 AppendAttributeAsDouble

This method adds a new attribute to an existing node. The value of the attribute has the data type Double. The name and value of the new attribute and the existing XML node are transferred to the method as input parameters. The method returns a reference to the newly added attribute.

Syntax

```
METHOD AppendAttributeAsDouble : SXmlAttribute
VAR_INPUT
    n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
    name : STRING;
END_VAR
VAR_INPUT
    value : LREAL;
END_VAR
```

Sample call:

```
objAttribute := fbXml.AppendAttributeAsDouble(objMachine, 'Name', 42.42);
```

5.2.1.6.4 AppendAttributeAsFloat

This method adds a new attribute to an existing node. The value of the attribute has the data type Float. The name and value of the new attribute and the existing XML node are transferred to the method as input parameters. The method returns a reference to the newly added attribute.

Syntax

```
METHOD AppendAttributeAsFloat : SXmlAttribute
VAR_INPUT
    n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
    name : STRING;
END_VAR
VAR_INPUT
    value : REAL;
END_VAR
```

Sample call:

```
objAttribute := fbXml.AppendAttributeAsFloat(objMachine, 'Name', 42.42);
```

5.2.1.6.5 AppendAttributeAsInt

This method adds a new attribute to an existing node. The value of the attribute has the data type Integer. The name and value of the new attribute and the existing XML node are transferred to the method as input parameters. The method returns a reference to the newly added attribute.

Syntax

```
METHOD AppendAttributeAsInt : SXmlAttribute
VAR_INPUT
    n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
    name : STRING;
END_VAR
VAR_INPUT
    value : DINT;
END_VAR
```

Sample call:

```
objAttribute := fbXml.AppendAttributeAsInt(objMachine, 'Name', 42);
```

5.2.1.6.6 AppendAttributeAsLint

This method adds a new attribute to an existing node. The value of the attribute has the data type Integer64. The name and value of the new attribute and the existing XML node are transferred to the method as input parameters. The method returns a reference to the newly added attribute.

Syntax

```
METHOD AppendAttributeAsLint : SXmlAttribute
VAR_INPUT
    n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
    name : STRING;
END_VAR
VAR_INPUT
    value : LINT;
END_VAR
```

Sample call:

```
objAttribute := fbXml.AppendAttributeAsLint(objMachine, 'Name', 42);
```

5.2.1.6.7 AppendAttributeAsUint

This method adds a new attribute to an existing node. The value of the attribute has the data type Unsigned Integer. The name and value of the new attribute and the existing XML node are transferred to the method as input parameters. The method returns a reference to the newly added attribute.

Syntax

```
METHOD AppendAttributeAsUint : SXmlAttribute
VAR_INPUT
    n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
    name : STRING;
END_VAR
VAR_INPUT
    value : UDINT;
END_VAR
```

Sample call:

```
objAttribute := fbXml.AppendAttributeAsUint(objMachine, 'Name', 42);
```

5.2.1.6.8 AppendAttributeAsUlint

This method adds a new attribute to an existing node. The value of the attribute has the data type Unsigned Integer64. The name and value of the new attribute and the existing XML node are transferred to the method as input parameters. The method returns a reference to the newly added attribute.

Syntax

```
METHOD AppendAttributeAsUlint : SXmlAttribute
VAR_INPUT
    n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
    name : STRING;
END_VAR
VAR_INPUT
    value : ULINT;
END_VAR
```

Sample call:

```
objAttribute := fbXml.AppendAttributeAsUlint(objMachine, 'Name', 42);
```

5.2.1.6.9 AppendAttributeCopy

This method adds a new attribute to an existing node. The name and value of the new attribute are copied from an existing attribute. The existing attribute is transferred to the method as input parameter.

Syntax

```
METHOD AppendAttributeCopy : SXmlAttribute
INPUT_VAR
  n : SXmlNode;
  copy : SXmlAttribute;
END_VAR
```

Sample call:

```
xmlNewAttribute := fbXml.AppendAttributeCopy(xmlNode, xmlExistingAttribute);
```

5.2.1.6.10 AppendChild

This method inserts a new node below an existing node. The value of the new node has the data type STRING. The name and value of the new node and a reference to the existing node are transferred to the method as input parameters. The method returns a reference to the newly added node. The input parameter cdata indicates whether the value of the node is to be encapsulated in a CDATA function block, so that certain special characters such as "<" and ">" are allowed as values.

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
```

Sample call:

```
xmlNewNode := fbXml.AppendChild(xmlExisting, 'Controller', 'CX5120', FALSE);
```

5.2.1.6.11 AppendChildAsBool

This method inserts a new node below an existing node. The value of the new node has the data type Boolean. The name and value of the new node and a reference to the existing node are transferred to the method as input parameters. The method returns a reference to the newly added node.

Syntax

```
METHOD AppendChildAsBool : SXmlNode
VAR_INPUT
  n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name : STRING;
END_VAR
VAR_INPUT
  value : BOOL;
END_VAR
```

Sample call:

```
xmlNewNode := fbXml.AppendChildAsBool(xmlExisting, 'SomeName', TRUE);
```

5.2.1.6.12 AppendChildAsDouble

This method inserts a new node below an existing node. The value of the new node has the data type Double. The name and value of the new node and a reference to the existing node are transferred to the method as input parameters. The method returns a reference to the newly added node.

Syntax

```
METHOD AppendChildAsDouble : SXmlNode
VAR_INPUT
    n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
    name : STRING;
END_VAR
VAR_INPUT
    value : LREAL;
END_VAR
```

Sample call:

```
xmlNewNode := fbXml.AppendChildAsDouble(xmlExisting, 'SomeName', 42.42);
```

5.2.1.6.13 AppendChildAsFloat

This method inserts a new node below an existing node. The value of the new node has the data type Float. The name and value of the new node and a reference to the existing node are transferred to the method as input parameters. The method returns a reference to the newly added node.

Syntax

```
METHOD AppendChildAsFloat : SXmlNode
VAR_INPUT
    n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
    name : STRING;
END_VAR
VAR_INPUT
    value : REAL;
END_VAR
```

Sample call:

```
xmlNewNode := fbXml.AppendChildAsFloat(xmlExisting, 'SomeName', 42.42);
```

5.2.1.6.14 AppendChildAsInt

This method inserts a new node below an existing node. The value of the new node has the data type Integer. The name and value of the new node and a reference to the existing node are transferred to the method as input parameters. The method returns a reference to the newly added node.

Syntax

```
METHOD AppendChildAsInt : SXmlNode
VAR_INPUT
    n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
    name : STRING;
END_VAR
VAR_INPUT
    value : DINT;
END_VAR
```

Sample call:

```
xmlNewNode := fbXml.AppendChildAsInt(xmlExisting, 'SomeName', 42);
```

5.2.1.6.15 AppendChildAsLint

This method inserts a new node below an existing node. The value of the new node has the data type Integer64. The name and value of the new node and a reference to the existing node are transferred to the method as input parameters. The method returns a reference to the newly added node.

Syntax

```
METHOD AppendChildAsLint : SXmlNode
VAR_INPUT
    n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
    name : STRING;
END_VAR
VAR_INPUT
    value : LINT;
END_VAR
```

Sample call:

```
xmlNewNode := fbXml.AppendChildAsLint(xmlExisting, 'SomeName', 42);
```

5.2.1.6.16 AppendChildAsUint

This method inserts a new node below an existing node. The value of the new node has the data type Unsigned Integer. The name and value of the new node and a reference to the existing node are transferred to the method as input parameters. The method returns a reference to the newly added node.

Syntax

```
METHOD AppendChildAsUint : SXmlNode
VAR_INPUT
    n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
    name : STRING;
END_VAR
VAR_INPUT
    value : UDINT;
END_VAR
```

Sample call:

```
xmlNewNode := fbXml.AppendChildAsUint(xmlExisting, 'SomeName', 42);
```

5.2.1.6.17 AppendChildAsUlint

This method inserts a new node below an existing node. The value of the new node has the data type Unsigned Integer64. The name and value of the new node and a reference to the existing node are transferred to the method as input parameters. The method returns a reference to the newly added node.

Syntax

```
METHOD AppendChildAsUlint : SXmlNode
VAR_INPUT
    n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
    name : STRING;
END_VAR
VAR_INPUT
    value : ULINT;
END_VAR
```

Sample call:

```
xmlNewNode := fbXml.AppendChildAsUlint(xmlExisting, 'SomeName', 42);
```

5.2.1.6.18 AppendCopy

This method inserts a new node below an existing node. The name and value of the new node are copied from an existing node. The references to the existing nodes are transferred to the method as input parameters. The method returns a reference to the newly added node.

Syntax

```
METHOD AppendCopy : SXmlNode
VAR_INPUT
    n : SXmlNode;
    copy : SXmlNode;
END_VAR
```

Sample call:

```
xmlNewNode := fbXml.AppendCopy(xmlParentNode, xmlExistingNode);
```

5.2.1.6.19 AppendNode

This method adds a new node to an existing node. The existing node and the name of the new node are transferred to the method as input parameters. The method returns a reference to the newly added node.

Syntax

```
METHOD AppendNode : SXmlNode
VAR_INPUT
    n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
    name : STRING;
END_VAR
```

Sample call:

```
objMachines := fbXml.AppendNode(objRoot, 'Machines');
```

5.2.1.6.20 Attributes

This method can be used to read the attribute of a given XML node. The XML node and the name of the attribute are transferred to the method as input parameters. After the method has been called, further methods have to be called, for example to read the value of the attribute, e.g. AttributeAsInt().

Syntax

```
METHOD Attribute : SXmlAttribute
VAR_INPUT
    n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
    name : STRING;
END_VAR
```

Sample call:

```
xmlMachine1Attribute := fbXml.Attribute(xmlMachine1, 'Type');
```

5.2.1.6.21 AttributeAsBool

This method returns the value of an attribute as data type Boolean. The attribute is transferred to the method as input parameter.

Syntax

```
METHOD AttributeAsBool : BOOL
VAR_INPUT
    a : SXmlAttribute;
END_VAR
```

Sample call:

```
bValue := fbXml.AttributeAsBool(xmlAttr);
```

5.2.1.6.22 AttributeAsDouble

This method returns the value of an attribute as data type Double. The attribute is transferred to the method as input parameter.

Syntax

```
METHOD AttributeAsDouble : LREAL
VAR_INPUT
    a : SXmlAttribute;
END_VAR
```

Sample call:

```
lrValue := fbXml.AttributeAsDouble(xmlAttr);
```

5.2.1.6.23 AttributeAsFloat

This method returns the value of an attribute as data type Float. The attribute is transferred to the method as input parameter.

Syntax

```
METHOD AttributeAsFloat : REAL
VAR_INPUT
    a : SXmlAttribute;
END_VAR
```

Sample call:

```
rValue := fbXml.AttributeAsFloat(xmlAttr);
```

5.2.1.6.24 AttributeAsInt

This method returns the value of an attribute as a data type Integer. The attribute is transferred to the method as input parameter.

Syntax

```
METHOD AttributeAsInt : DINT
VAR_INPUT
    a : SXmlAttribute;
END_VAR
```

Sample call:

```
nValue := fbXml.AttributeAsInt(xmlAttr);
```

5.2.1.6.25 AttributeAsLint

This method returns the value of an attribute as a data type Integer64. The attribute is transferred to the method as input parameter.

Syntax

```
METHOD AttributeAsLint : LINT
VAR_INPUT
    a : SXmlAttribute;
END_VAR
```

Sample call:

```
nValue := fbXml.AttributeAsLint(xmlAttr);
```

5.2.1.6.26 AttributeAsUint

This method returns the value of an attribute as data type Unsigned Integer. The attribute is transferred to the method as input parameter.

Syntax

```
METHOD AttributeAsUint : UDINT
VAR_INPUT
    a : SXmlAttribute;
END_VAR
```

Sample call:

```
nValue := fbXml.AttributeAsUint(xmlAttr);
```

5.2.1.6.27 AttributeAsUlint

This method returns the value of an attribute as data type Unsigned Integer64. The attribute is transferred to the method as input parameter.

Syntax

```
METHOD AttributeAsUlint : ULINT
VAR_INPUT
    a : SXmlAttribute;
END_VAR
```

Sample call:

```
nValue := fbXml.AttributeAsUlint(xmlAttr);
```

5.2.1.6.28 AttributeBegin

This method returns an iterator over all attributes of an XML node. The XML node is transferred to the method as input parameter.

Syntax

```
METHOD AttributeBegin : SXmlIterator
VAR_INPUT
    n : SXmlNode;
END_VAR
```

Sample call:

```
xmlIterator := fbXml.AttributeBegin(xmlNode);
WHILE NOT fbXml.IsEnd(xmlIterator) DO
    xmlAttr := fbXml.AttributeFromIterator(xmlIterator);
    nAttrValue := fbXml.AttributeAsInt(xmlAttr);
    xmlIterator := fbXml.Next(xmlIterator);
END_WHILE
```

5.2.1.6.29 AttributeFromIterator

This method converts the current position of an iterator to an XML attribute object. The iterator is transferred to the method as input parameter.

Syntax

```
METHOD AttributeFromIterator : SXmlAttribute
VAR_INPUT
    it : SXmlIterator;
END_VAR
```

Sample call:

```
xmlIterator := fbXml.AttributeBegin(xmlNode);
WHILE NOT fbXml.IsEnd(xmlIterator) DO
    xmlAttr := fbXml.AttributeFromIterator(xmlIterator);
    nAttrValue := fbXml.AttributeAsInt(xmlAttr);
    xmlIterator := fbXml.Next(xmlIterator);
END_WHILE
```

5.2.1.6.30 AttributeName

This method returns the name of a given attribute. The attribute is transferred to the method as input parameter.

Syntax

```
METHOD AttributeName : STRING
VAR_INPUT
    a : SXmlAttribute;
END_VAR
```

Sample call:

```
sName := fbXml.AttributeName(xmlAttr);
```

5.2.1.6.31 Attributes

This method is used to navigate through the DOM and returns an iterator for all attributes found at an XML node. The iterator can then be used for further navigation through the elements that were found. The node and a reference to the iterator are transferred to the method as input parameters.

Syntax

```
METHOD Attributes : SXmlAttribute
VAR_INPUT
    n : SXmlNode;
    it : REFERENCE TO SXmlIterator;
END_VAR
```

Sample call:

```
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
```

5.2.1.6.32 AttributeText

This method returns the text of a given attribute. The attribute is transferred to the method as input parameter.

Syntax

```
METHOD AttributeText : STRING(255)
VAR_INPUT
    a : SXmlAttribute;
END_VAR
```

Sample call:

```
sText := fbXml.AttributeText(xmlAttr);
```

5.2.1.6.33 Begin

This method returns an iterator over all child elements of an XML node, always starting from the first child element. The XML node is transferred to the method as input parameter.

Syntax

```
METHOD Begin : SXmlIterator
VAR_INPUT
    n : SXmlNode;
END_VAR
```

Sample call:

```
xmlIterator := fbXml.Begin(xmlNode);
WHILE NOT fbXml.IsEnd(xmlIterator) DO
    xmlNodeRef := fbXml.Node(xmlIterator);
    xmlNodeValue := fbXml.NodeText(xmlNodeRef);
    xmlIterator := fbXml.Next(xmlIterator);
END_WHILE
```

5.2.1.6.34 BeginByName

This method returns an iterator over all child elements of an XML node, starting at a particular element. The XML node is transferred to the method as input parameter.

Syntax

```
METHOD BeginByName : SXmlIterator
VAR_INPUT
    n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
    name : STRING;
END_VAR
```

Sample call:

```
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
```

5.2.1.6.35 Child

This method is used to navigate through the DOM. It returns a reference to the (first) child element of the current node. The start node is transferred to the method as input parameter.

Syntax

```
METHOD ChildByName : SXmlNode
VAR_INPUT
    n : SXmlNode;
END_VAR
```

Sample call:

```
xmlChild := fbXml.Child(xmlNode);
```

5.2.1.6.36 ChildByAttribute

This method is used to navigate through the DOM. It returns a reference to a child element in the XML document. The start node and the name and value of the attribute are transferred to the method as input parameters.

Syntax

```
METHOD ChildByAttribute : SXmlNode
VAR_INPUT
    n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
```

```

    attr : STRING;
    value : STRING;
END_VAR

```

Sample call:

```
xmlMachine1 := fbXml.ChildByAttribute(xmlMachines, 'Type', '1');
```

5.2.1.6.37 ChildByAttributeAndName

This method is used to navigate through the DOM. It returns a reference to a child element in the XML document. The start node, the name and value of the attribute, and the name of the child element are transferred to the method as input parameters.

Syntax

```

METHOD ChildByAttributeAndName : SXmlNode
VAR_INPUT
    n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
    attr : STRING;
    value : STRING;
    child : STRING;
END_VAR

```

Sample call:

```
xmlMachine2 := fbXml.ChildByAttributeAndName(xmlMachines, 'Type', '2', 'Machine');
```

5.2.1.6.38 ChildByName

This method is used to navigate through the DOM. It returns a reference to a child element in the XML document. The start node and the name of the element to be returned are transferred to the method as input parameters.

Syntax

```

METHOD ChildByName : SXmlNode
VAR_INPUT
    n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
    name : STRING;
END_VAR

```

Sample call:

```
xmlMachines := fbXml.ChildByName(xmlDoc, 'Machines');
```

5.2.1.6.39 Children

This method is used to navigate through the DOM. It returns an iterator for several child elements found in the XML document. The iterator can then be used for further navigation through the elements that were found. The start node and a reference to the iterator are transferred to the method as input parameters.

Syntax

```

METHOD Children : SXmlNode
VAR_INPUT
    n : SXmlNode;
    it : REFERENCE TO SXmlIterator;
END_VAR

```

Sample call:

```

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

```

5.2.1.6.40 ChildrenByName

This method is used to navigate through the DOM. It returns an iterator for several child elements found in the XML document. The iterator can then be used for further navigation through the elements that were found. The start node, the name of the child elements to be found and a reference to the iterator are transferred to the method as input parameters.

Syntax

```

METHOD ChildrenByName : SXmlNode
VAR_INPUT
    n : SXmlNode;
    it : REFERENCE TO SXmlIterator;
END_VAR
VAR_IN_OUT CONSTANT
    name : STRING;
END_VAR

```

Sample call:

```

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

```

5.2.1.6.41 Compare

This method checks two iterators for equality.

Syntax

```

METHOD Compare : BOOL
VAR_INPUT
    it1 : SXmlIterator;
    it2 : SXmlIterator;
END_VAR

```

Sample call:

```

bResult := fbXml.Compare(xmlIt1, xmlIt2);

```

5.2.1.6.42 CopyAttributeText

This method reads the value of an XML attribute and writes it to a variable of data type String. The XML attribute, the target variable and the length to be written are transferred to the method as input parameters. The method returns the actual size.

Syntax

```

METHOD CopyAttributeText : UDINT
VAR_INPUT
    a : SXmlAttribute;
END_VAR
VAR_IN_OUT CONSTANT
    sXml : STRING;
END_VAR
VAR_INPUT
    nXml : UDINT;
END_VAR

```

Sample call:

```

nLength := fbXml.CopyAttributeText(xmlAttr, sTarget, SIZEOF(sTarget));

```

5.2.1.6.43 CopyDocument

This method copies the contents of the DOM memory into a variable of the data type String. The length to be written and the variable into which the resulting string is to be written are transferred to the method as input parameters. The method returns the actually written length. Note that the size of the string variable is at least equal to the size of the XML document in the DOM.

Syntax

```
METHOD CopyDocument : UDINT
VAR_IN_OUT CONSTANT
    sXml : STRING;
END_VAR
VAR_INPUT
    nXml : UDINT;
END_VAR
```

Sample call:

```
nLength := fbXml.CopyDocument(sTarget, SIZEOF(sTarget));
```

5.2.1.6.44 CopyNodeText

This method reads the value of an XML node and writes it to a variable of data type String. The XML node, the target variable and the length to be written are transferred to the method as input parameters. The method returns the actual size.

Syntax

```
METHOD CopyNodeText : UDINT
VAR_INPUT
    n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
    sXml : STRING;
END_VAR
VAR_INPUT
    nXml : UDINT;
END_VAR
```

Sample call:

```
nLength := fbXml.CopyNodeText(xmlNode, sTarget, SIZEOF(sTarget));
```

5.2.1.6.45 CopyNodeXml

This method reads the XML structure of an XML node and writes it to a variable of data type String. The XML node, the target variable and the length to be written are transferred to the method as input parameters. The method returns the actual size.

Syntax

```
METHOD CopyNodeXml : UDINT
VAR_INPUT
    a : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
    sXml : STRING;
END_VAR
VAR_INPUT
    nXml : UDINT;
END_VAR
```

Sample call:

```
nLength := fbXml.CopyNodeXml(xmlNode, sTarget, SIZEOF(sTarget));
```

5.2.1.6.46 FirstNodeByPath

This method navigates through an XML document using a path that was transferred to the method. The path and the start node are transferred to the method as input parameters. The path is specified with "/" as separator. The method returns a reference to the XML node that was found.

Syntax

```
METHOD FirstNodeByPath : SXmlNode
VAR_INPUT
    n : SXmlNode;
    path : STRING;
END_VAR
```

Sample call:

```
xmlFoundNode := fbXml.FirstNodeByPath(xmlStartNode, 'Level1/Level2/Level3');
```

5.2.1.6.47 GetAttributeTextLength

This method returns the length of the value of an XML attribute. The XML attribute is transferred to the method as input parameter.

Syntax

```
METHOD GetAttributeTextLength : UDINT
VAR_INPUT
    a : SXmlAttribute;
END_VAR
```

Sample call:

```
nLength := fbXml.GetAttributeTextLength(xmlAttr);
```

5.2.1.6.48 GetDocumentLength

This method returns the length of an XML document in bytes.

Syntax

```
METHOD GetDocumentLength : UDINT
```

Sample call:

```
nLength := fbXml.GetDocumentLength();
```

5.2.1.6.49 GetDocumentNode

This method returns the root node of an XML document. This is not the same as the first XML node in the document (the method GetRootNode() should be used for this). The method can also be used to create an empty XML document in the DOM.

Syntax

```
METHOD GetDocumentNode : SXmlNode
```

Sample call:

```
objRoot := fbXml.GetDocumentNode();
```

5.2.1.6.50 GetNodeTextLength

This method returns the length of the value of an XML node. The XML node is transferred to the method as input parameter.

Syntax

```
METHOD GetNodeTextLength : UDINT
VAR_INPUT
    n : SXmlNode;
END_VAR
```

Sample call:

```
nLength := fbXml.GetNodeTextLength(xmlNode);
```

5.2.1.6.51 GetNodeXmlLength

This method returns the length of the XML structure of an XML node. The XML node is transferred to the method as input parameter.

Syntax

```
METHOD GetNodeXmlLength : UDINT
VAR_INPUT
    n : SXmlNode;
END_VAR
```

Sample call:

```
nLength := fbXml.GetNodeXmlLength(xmlNode);
```

5.2.1.6.52 GetRootNode

This method returns a reference to the first XML node in the XML document.

Syntax

```
METHOD GetRootNode : SXmlNode
```

Sample call:

```
xmlRootNode := fbXml.GetRootNode();
```

5.2.1.6.53 InsertAttributeCopy

This method adds an attribute to an XML node. The name and value of an existing attribute are copied. The attribute can be placed at a specific position. The XML node, the position and a reference to the existing attribute object are transferred to the method as input parameters. The method returns a reference to the newly added attribute.

Syntax

```
METHOD InsertAttributeCopy : SXmlAttribute
VAR_INPUT
    n : SXmlNode;
    before : SXmlAttribute;
    copy : SXmlAttribute;
END_VAR
```

Sample call:

```
xmlNewAttr := fbXml.InsertAttributeCopy(xmlNode, xmlBeforeAttr, xmlCopyAttr);
```

5.2.1.6.54 InsertAttribute

This method adds an attribute to an XML node. The attribute can be placed at a specific position. The XML node and the position and name of the new attribute are transferred to the method as input parameters. The method returns a reference to the newly added attribute. A value for the attribute can then be entered using the SetAttribute() method, for example.

Syntax

```

METHOD InsertAttribute : SXmlAttribute
VAR_INPUT
    n : SXmlNode;
    before : SXmlAttribute;
END_VAR
VAR_IN_OUT CONSTANT
    name : STRING;
END_VAR

```

Sample call:

```
xmlNewAttr := fbXml.InsertAttribute(xmlNode, xmlBeforeAttr, 'SomeName');
```

5.2.1.6.55 InsertChild

This method adds a node to an existing XML node. The new node can be placed at a specific location. The existing XML node and the position and name of the new node are transferred to the method as input parameters. The method returns a reference to the newly added node. A value for the node can then be entered using the SetChild() method, for example.

Syntax

```

METHOD InsertChild : SXmlNode
VAR_INPUT
    n : SXmlNode;
    before : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
    name : STRING;
END_VAR

```

Sample call:

```
xmlNewNode := fbXml.InsertChild(xmlNode, xmlBeforeNode, 'SomeName');
```

5.2.1.6.56 InsertCopy

This method adds a new node to an existing XML node and copies an existing node. The new node can be placed anywhere in the existing node. The XML node, the position and a reference to the existing node object are transferred to the method as input parameters. The method returns a reference to the newly added node.

Syntax

```

METHOD InsertCopy : SXmlNode
VAR_INPUT
    n : SXmlNode;
    before : SXmlNode;
    copy : SXmlNode;
END_VAR

```

Sample call:

```
xmlNewNode := fbXml.InsertCopy(xmlNode, xmlBeforeNode, xmlCopyNode);
```

5.2.1.6.57 IsEnd

This method checks whether a given XML iterator is at the end of the iteration that is to be performed.

Syntax

```

METHOD IsEnd : BOOL
VAR_INPUT
    it : SXmlIterator;
END_VAR

```

Sample call:

```
xmlIterator := fbXml.Begin(xmlNode);
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlNodeRef := fbXml.Node(xmlIterator);
  xmlNodeValue := fbXml.NodeText(xmlNodeRef);
  xmlIterator := fbXml.Next(xmlIterator);
END_WHILE
```

5.2.1.6.58 LoadDocumentFromFile

This method loads an XML document from a file. The absolute path to the file is transferred to the method as input parameter.

A rising edge on the input parameter bExec triggers the loading procedure. The asynchronous process is terminated as soon as the reference bExec is set back to FALSE from the method. When the process ends, the return value of the method indicates for one call whether the loading of the file was successful (TRUE) or failed (FALSE).

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
```

Sample call:

```
IF bLoad THEN
  bLoaded := fbXml.LoadDocumentFromFile('C:\Test.xml', bLoad);
END_IF
```

5.2.1.6.59 NewDocument

This method creates an empty XML document in the DOM memory.

Syntax

```
METHOD NewDocument : BOOL
```

Sample call:

```
fbXml.NewDocument();
```

5.2.1.6.60 Next

This method sets an XML iterator for the next object that is to be processed.

Syntax

```
METHOD Next : SXmlIterator
VAR_INPUT
  it : SXmlIterator;
END_VAR
```

Sample call:

```
xmlIterator := fbXml.Begin(xmlNode);
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlNodeRef := fbXml.Node(xmlIterator);
  xmlNodeValue := fbXml.NodeText(xmlNodeRef);
  xmlIterator := fbXml.Next(xmlIterator);
END_WHILE
```

5.2.1.6.61 NextAttribute

This method returns the next attribute for a given XML attribute.

Syntax

```
METHOD NextAttribute : SXmlAttribute
VAR_INPUT
    a : SXmlAttribute;
END_VAR
```

Sample call:

```
xmlNextAttr := fbXml.NextAttribute(xmlAttr);
```

5.2.1.6.62 NextByName

This method sets an XML iterator for the next object that is to be processed, which is identified by its name.

Syntax

```
METHOD NextByName : SXmlIterator
VAR_INPUT
    it : SXmlIterator;
END_VAR
VAR_IN_OUT CONSTANT
    name : STRING;
END_VAR
```

Sample call:

```
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
```

5.2.1.6.63 NextSibling

This method returns the next direct node for a given XML node at the same XML level.

Syntax

```
METHOD NextSibling : SXmlNode
VAR_INPUT
    n : SXmlNode;
END_VAR
```

Sample call:

```
xmlSibling := fbXml.NextSibling(xmlNode);
```

5.2.1.6.64 NextSiblingByName

This method returns the next direct node for a given XML node with a particular name at the same XML level.

Syntax

```
METHOD NextSiblingByName : SXmlNode
VAR_INPUT
    n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
    name : STRING;
END_VAR
```

Sample call:

```
xmlSibling := fbXml.NextSibling(xmlNode, 'SomeName');
```

5.2.1.6.65 Node

This method is used in conjunction with an iterator to navigate through the DOM. The iterator is transferred to the method as input parameter. The method then returns the current XML node as return value.

Syntax

```
METHOD Node : SXmlNode
VAR_INPUT
    it : SXmlIterator;
END_VAR
```

Sample call:

```
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
```

5.2.1.6.66 NodeAsBool

This method returns the text of an XML node as data type Boolean. The XML node is transferred to the method as input parameter.

Syntax

```
METHOD NodeAsBool : BOOL
VAR_INPUT
    n : SXmlNode;
END_VAR
```

Sample call:

```
bXmlNode:= fbXml.NodeAsBool(xmlMachine1);
```

5.2.1.6.67 NodeAsDouble

This method returns the text of an XML node as data type Double. The XML node is transferred to the method as input parameter.

Syntax

```
METHOD NodeAsDouble : LREAL
VAR_INPUT
    n : SXmlNode;
END_VAR
```

Sample call:

```
lrXmlNode:= fbXml.NodeAsDouble(xmlMachine1);
```

5.2.1.6.68 NodeAsFloat

This method returns the text of an XML node as data type Float. The XML node is transferred to the method as input parameter.

Syntax

```
METHOD NodeAsFloat : REAL
VAR_INPUT
    n : SXmlNode;
END_VAR
```

Sample call:

```
rXmlNode:= fbXml.NodeAsFloat(xmlMachine1);
```

5.2.1.6.69 NodeAsInt

This method returns the text of an XML node as a data type Integer. The XML node is transferred to the method as input parameter.

Syntax

```
METHOD NodeAsInt : DINT
VAR_INPUT
    n : SXmlNode;
END_VAR
```

Sample call:

```
nXmlNode:= fbXml.NodeAsInt(xmlMachine1);
```

5.2.1.6.70 NodeAsLint

This method returns the text of an XML node as a data type Integer64. The XML node is transferred to the method as input parameter.

Syntax

```
METHOD NodeAsLint : LINT
VAR_INPUT
    n : SXmlNode;
END_VAR
```

Sample call:

```
nXmlNode:= fbXml.NodeAsLint(xmlMachine1);
```

5.2.1.6.71 NodeAsUInt

This method returns the text of an XML node as data type Unsigned Integer. The XML node is transferred to the method as input parameter.

Syntax

```
METHOD NodeAsUInt : UDINT
VAR_INPUT
    n : SXmlNode;
END_VAR
```

Sample call:

```
nXmlNode:= fbXml.NodeAsUInt(xmlMachine1);
```

5.2.1.6.72 NodeAsUlint

This method returns the text of an XML node as data type Unsigned Integer64. The XML node is transferred to the method as input parameter.

Syntax

```
METHOD NodeAsUlint : ULINT
VAR_INPUT
    n : SXmlNode;
END_VAR
```

Sample call:

```
nXmlNode:= fbXml.NodeAsUlint(xmlMachine1);
```

5.2.1.6.73 NodeName

This method returns the name of an XML node. A reference to the XML node is transferred to the method as input parameter.

Syntax

```
METHOD NodeName : STRING
VAR_INPUT
    n : SXmlNode;
END_VAR
```

Sample call:

```
sNodeName := fbXml.NodeName(xmlMachine1);
```

5.2.1.6.74 NodeText

This method returns the text of an XML node. The XML node is transferred to the method as input parameter.

Syntax

```
METHOD NodeText : STRING(255)
VAR_INPUT
    n : SXmlNode;
END_VAR
```

Sample call:

```
sMachine1Name := fbXml.NodeText(xmlMachine1);
```

5.2.1.6.75 ParseDocument

This method loads an XML document into the DOM memory for further processing. The XML document exists as a string and is transferred to the method as input parameter. A reference to the XML document in the DOM is returned to the caller.

Syntax

```
METHOD ParseDocument : SXmlNode
VAR_IN_OUT CONSTANT
    sXml : STRING;
END_VAR
```

Sample call:

```
xmlDoc := fbXml.ParseDocument(sXmlToParse);
```

5.2.1.6.76 RemoveChild

This method removes an XML child node from a given XML node. The two XML nodes are transferred to the method as input parameters. The method returns TRUE if the operation was successful and the XML node was removed.

Syntax

```
METHOD RemoveChild : BOOL
VAR_INPUT
    n : SXmlNode;
    child : SXmlNode;
END_VAR
```

Sample call:

```
bRemoved := fbXml.RemoveChild(xmlParent, xmlChild);
```

5.2.1.6.77 RemoveChildByName

This method removes an XML child node from a given XML node. The node to be removed is addressed by its name. If there is more than one child node, the last child node is removed. The method returns TRUE if the operation was successful and the XML node was removed.

Syntax

```
METHOD RemoveChildByName : BOOL
VAR_INPUT
    n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
    name : STRING;
END_VAR
```

Sample call:

```
bRemoved := fbXml.RemoveChildByName(xmlParent, 'SomeName');
```

5.2.1.6.78 SaveDocumentToFile

This method saves the current XML document in a file. The absolute path to the file is transferred to the method as input parameter.

A rising edge at the input parameter bExec triggers the saving procedure. The asynchronous process is terminated as soon as the reference bExec is set back to FALSE from the method. When the process ends, the return value of the method indicates for one call whether saving of the file was successful (TRUE) or failed (FALSE).

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
```

Sample call:

```
IF bSave THEN
    bSaved = fbXml.SaveDocumentToFile('C:\Test.xml', bSave);
END_IF
```

5.2.1.6.79 SetAttribute

This method sets the value of an attribute. The value has the data type String.

Syntax

```
METHOD SetAttribute : SXmlAttribute
VAR_INPUT
    a : SXmlAttribute;
END_VAR
VAR_IN_OUT CONSTANT
    value : STRING;
END_VAR
```

Sample call:

```
xmlAttr := fbXml.SetAttribute(xmlExistingAttr, 'Test');
```

5.2.1.6.80 SetAttributeAsBool

This method sets the value of an attribute. The value has the data type Boolean.

Syntax

```
METHOD SetAttributeAsBool : SXmlAttribute
VAR_INPUT
    a : SXmlAttribute;
    value : BOOL;
END_VAR
```

Sample call:

```
xmlAttr := fbXml.SetAttributeAsBool(xmlExistingAttr, TRUE);
```

5.2.1.6.81 SetAttributeAsDouble

This method sets the value of an attribute. The value here has the data type Double.

Syntax

```
METHOD SetAttributeAsDouble : SXmlAttribute
VAR_INPUT
    a : SXmlAttribute;
    value : LREAL;
END_VAR
```

Sample call:

```
xmlAttr := fbXml.SetAttributeAsDouble(xmlExistingAttr, 42.42);
```

5.2.1.6.82 SetAttributeAsFloat

This method sets the value of an attribute. The value has the data type Float.

Syntax

```
METHOD SetAttributeAsFloat : SXmlAttribute
VAR_INPUT
    a : SXmlAttribute;
    value : REAL;
END_VAR
```

Sample call:

```
xmlAttr := fbXml.SetAttributeAsFloat(xmlExistingAttr, 42.42);
```

5.2.1.6.83 SetAttributeAsInt

This method sets the value of an attribute. The value has the data type Integer.

Syntax

```
METHOD SetAttributeAsInt : SXmlAttribute
VAR_INPUT
    a : SXmlAttribute;
    value : DINT;
END_VAR
```

Sample call:

```
xmlAttr := fbXml.SetAttributeAsInt(xmlExistingAttr, 42);
```

5.2.1.6.84 SetAttributeAsLint

This method sets the value of an attribute. The value has the data type Integer64.

Syntax

```
METHOD SetAttributeAsLint : SXmlAttribute
VAR_INPUT
  a : SXmlAttribute;
  value : LINT;
END_VAR
```

Sample call:

```
xmlAttr := fbXml.SetAttributeAsLint(xmlExistingAttr, 42);
```

5.2.1.6.85 SetAttributeAsUint

This method sets the value of an attribute. The value has the data type Unsigned Integer.

Syntax

```
METHOD SetAttributeAsUint : SXmlAttribute
VAR_INPUT
  a : SXmlAttribute;
  value : UDINT;
END_VAR
```

Sample call:

```
xmlAttr := fbXml.SetAttributeAsUint(xmlExistingAttr, 42);
```

5.2.1.6.86 SetAttributeAsUlint

This method sets the value of an attribute. The value has the data type Unsigned Integer64.

Syntax

```
METHOD SetAttributeAsUlint : SXmlAttribute
VAR_INPUT
  a : SXmlAttribute;
  value : ULINT;
END_VAR
```

Sample call:

```
xmlAttr := fbXml.SetAttributeAsUlint(xmlExistingAttr, 42);
```

5.2.1.6.87 SetChild

This method sets the value of an XML node. The value is transferred to the method as input parameter of data type String. The input parameter cdata indicates whether the value of the node is to be encapsulated in a CDATA function block, so that certain special characters such as "<" and ">" are allowed as values.

Syntax

```
METHOD SetChild : SXmlNode
VAR_INPUT
  n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  value : STRING;
END_VAR
VAR_INPUT
  cdata : BOOL;
END_VAR
```

Sample call:

```
xmlNode := fbXml.SetChild(xmlExistingNode, 'SomeText', FALSE);
```

5.2.1.6.88 SetChildAsBool

This method sets the value of an XML node. The value is transferred to the method as input parameter of data type Boolean.

Syntax

```
METHOD SetChildAsBool : SXmlNode
VAR_INPUT
    n : SXmlNode;
    value : BOOL;
END_VAR
```

Sample call:

```
xmlNode := fbXml.SetChild(xmlExistingNode, TRUE);
```

5.2.1.6.89 SetChildAsDouble

This method sets the value of an XML node. The value is transferred to the method as input parameter of data type Double.

Syntax

```
METHOD SetChildAsDouble : SXmlNode
VAR_INPUT
    n : SXmlNode;
    value : LREAL;
END_VAR
```

Sample call:

```
xmlNode := fbXml.SetChildAsDouble(xmlExistingNode, 42.42);
```

5.2.1.6.90 SetChildAsFloat

This method sets the value of an XML node. The value is transferred to the method as input parameter of data type Float.

Syntax

```
METHOD SetChildAsFloat : SXmlNode
VAR_INPUT
    n : SXmlNode;
    value : REAL;
END_VAR
```

Sample call:

```
xmlNode := fbXml.SetChildAsFloat(xmlExistingNode, 42.42);
```

5.2.1.6.91 SetChildAsInt

This method sets the value of an XML node. The value is transferred to the method as input parameter of data type Integer.

Syntax

```
METHOD SetChildAsInt : SXmlNode
VAR_INPUT
    n : SXmlNode;
    value : DINT;
END_VAR
```

Sample call:

```
xmlNode := fbXml.SetChildAsInt(xmlExistingNode, 42);
```

5.2.1.6.92 SetChildAsLint

This method sets the value of an XML node. The value is transferred to the method as input parameter of data type Integer64.

Syntax

```
METHOD SetChildAsLint : SXmlNode
VAR_INPUT
    n : SXmlNode;
    value : LINT;
END_VAR
```

Sample call:

```
xmlNode := fbXml.SetChildAsLint(xmlExistingNode, 42);
```

5.2.1.6.93 SetChildAsUint

This method sets the value of an XML node. The value is transferred to the method as input parameter of data type Unsigned Integer.

Syntax

```
METHOD SetChildAsUint : SXmlNode
VAR_INPUT
    n : SXmlNode;
    value : UDINT;
END_VAR
```

Sample call:

```
xmlNode := fbXml.SetChildAsUint(xmlExistingNode, 42);
```

5.2.1.6.94 SetChildAsUlint

This method sets the value of an XML node. The value is transferred to the method as input parameter of data type Unsigned Integer64.

Syntax

```
METHOD SetChildAsUlint : SXmlNode
VAR_INPUT
    n : SXmlNode;
    value : ULINT;
END_VAR
```

Sample call:

```
xmlNode := fbXml.SetChildAsUlint(xmlExistingNode, 42);
```

5.2.1.7 FB_JwtEncode

The function block enables the creation and signing of a 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

```

Inputs

Name	Type	Description
bExecute	BOOL	A rising edge activates processing of the function block.
sHeaderAlg	STRING(46)	The algorithm to be used for the JWT header, e.g. RS256.
sPayload	STRING(1023)	The JWT payload to be used.
sKeyFilePath	STRING(511)	Path to the private key to be used for the signature of the JWT.
tTimeout	TIME	ADS timeout, which is used internally for file access to the private key.
pKey	PVOID	Buffer for the private key to be read.
nKeySize	UDINT	Maximum size of the buffer.
sJwt	STRING [► 46]	Contains the fully coded and signed JWT after the function block has been processed.
nJwtSize	UDINT [► 47]	Size of the generated JWT including zero termination.

Outputs

Name	Type	Description
bBusy	BOOL	Is TRUE as long as processing of the function block is in progress.
bError	BOOL	Becomes TRUE as soon as an error situation occurs.
hrErrorCode	HRESULT	Returns an error code if the bError output is set. An explanation of the possible error codes [► 156] can be found in the Appendix.
initStatus	HRESULT	Returns an error code in case of a failed initialization of the function block.

Requirements

TwinCAT version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4024.4	x86, x64, ARM	Tc3_JsonXml 3.3.6.0

5.2.2 Interfaces

5.2.2.1 ITcJsonSaxHandler

5.2.2.1.1 OnBool

This callback method is triggered if a value of the data type BOOL was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnBool : HRESULT
VAR_INPUT
    value : BOOL;
END_VAR
```

5.2.2.1.2 OnDint

This callback method is triggered if a value of the data type DINT was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnDint : HRESULT
VAR_INPUT
    value : DINT;
END_VAR
```

5.2.2.1.3 OnEndArray

This callback method is triggered if a square closing bracket, which corresponds to the JSON synonym for an ending array, was found at the position of the SAX reader. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnEndArray : HRESULT
```

5.2.2.1.4 OnEndObject

This callback method is triggered if a curly closing bracket, which corresponds to the JSON synonym for an ending object, was found at the position of the SAX reader. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnEndObject : HRESULT
```

5.2.2.1.5 OnKey

This callback method is triggered if a property was found at the position of the SAX reader. The property name lies on the input/output parameter key and its length on the input parameter len. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnKey : HRESULT
VAR_IN_OUT CONSTANT
    key : STRING;
END_VAR
VAR_INPUT
    len : UDINT;
END_VAR
```

5.2.2.1.6 OnLint

This callback method is triggered if a value of the data type LINT was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnLint : HRESULT
VAR_INPUT
    value : LINT;
END_VAR
```

5.2.2.1.7 OnLreal

This callback method is triggered if a value of the data type LREAL was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnLreal : HRESULT
VAR_INPUT
    value : LREAL;
END_VAR
```

5.2.2.1.8 OnNull

This callback method is triggered if a NULL value was found at the position of the SAX reader. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnNull : HRESULT
```

5.2.2.1.9 OnStartArray

This callback method is triggered if a square opening bracket, which corresponds to the JSON synonym for a starting array, was found at the position of the SAX reader. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnStartArray : HRESULT
```

5.2.2.1.10 OnStartObject

This callback method is triggered if a curly opening bracket, which corresponds to the JSON synonym for a starting object, was found at the position of the SAX reader. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnStartObject : HRESULT
```

5.2.2.1.11 OnString

This callback method is triggered if a value of the data type STRING was found at the position of the SAX reader. The In/Out parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnString : HRESULT
VAR_IN_OUT CONSTANT
    value : STRING;
END_VAR
```

```
VAR_INPUT
    len : UDINT;
END_VAR
```

5.2.2.1.12 OnUdint

This callback method is triggered if a value of the data type UDINT was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnUdint : HRESULT
VAR_INPUT
    value : UDINT;
END_VAR
```

5.2.2.1.13 OnUlint

This callback method is triggered if a value of the data type ULINT was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnUlint : HRESULT
VAR_INPUT
    value : ULINT;
END_VAR
```

5.2.2.2 ITcJsonSaxValues

5.2.2.2.1 OnBoolValue

This callback method is triggered if a value of the data type BOOL was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnBoolValue : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
    value : BOOL;
END_VAR
```

5.2.2.2.2 OnDintValue

This callback method is triggered if a value of the data type DINT was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnDintValue : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
    value : DINT;
END_VAR
```

5.2.2.2.3 OnLintValue

This callback method is triggered if a value of the data type LINT was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnLintValue : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
    value : LINT;
END_VAR
```

5.2.2.2.4 OnLrealValue

This callback method is triggered if a value of the data type LREAL was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnLrealValue : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
    value : LREAL;
END_VAR
```

5.2.2.2.5 OnNullValue

This callback method is triggered if a NULL value was found at the position of the SAX reader. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnNull : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
END_VAR
```

5.2.2.2.6 OnStringValue

This callback method is triggered if a value of the data type STRING was found at the position of the SAX reader. The input/output parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

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

This callback method is triggered if a value of the data type UDINT was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnUdintValue : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
    value : UDINT;
END_VAR
```

5.2.2.2.8 OnUlintValue

This callback method is triggered if a value of the data type ULINT was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnUlintValue : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
    value : ULINT;
END_VAR
```

6 Samples

The following samples illustrate the communication with an MQTT broker. Messages are sent and received.

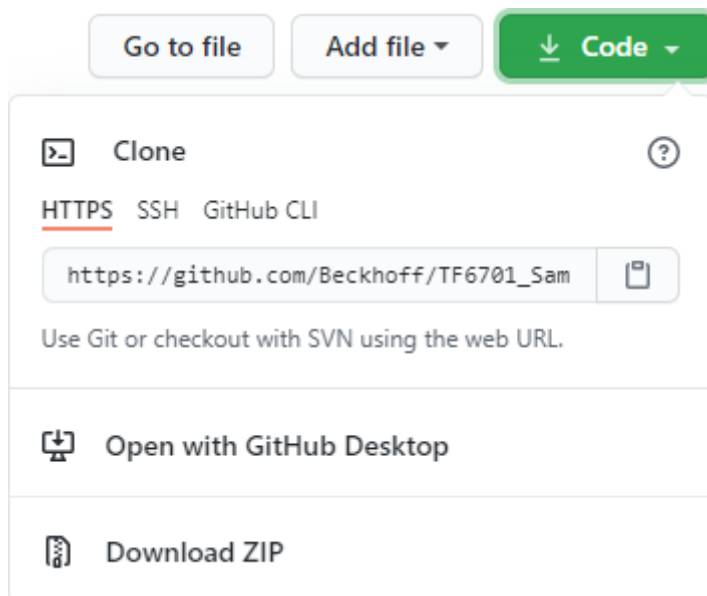
There are two different implementation options, which can be chosen based on purely subjective criteria. The two options are compared in the first two samples.

Overview

Sample	Link	Description
1	lotMqttSampleUsingQueue [► 135]	MQTT communication based on a messages queue
2	lotMqttSampleUsingCallback [► 137]	MQTT communication based on a callback method
3	lotMqttSampleTlsPsk [► 139]	MQTT communication via a secure TLS connection and PSK (PreSharedKey)
4	lotMqttSampleTlsCa [► 140]	MQTT communication via a secure TLS connection and CA (certificate authority) certificate
5	lotMqttSampleAwsIoT [► 140]	MQTT communication with AWS IoT
6	lotMqttSampleAzureIoT Hub [► 142]	MQTT communication with the Microsoft Azure IoT Hub
7	lotMqttSampleIbmWatsonIoT [► 146]	MQTT communication with IBM Watson IoT
8	lotMqttSampleMathworksThingspeak [► 147]	MQTT communication with the ThingSpeak IoT platform from MathWorks

Downloads

Sample code and configurations for this product can be obtained from the corresponding repository on GitHub: https://www.github.com/Beckhoff/TF6701_Samples. There you have the option to clone the repository or download a ZIP file containing the sample.



6.1 lotMqttSampleUsingQueue

Sample for MQTT communication via a message queue

This sample illustrates the communication with an MQTT broker. Messages are sent (publish mode) and received. Receiving of messages involves two steps. First, a general decision is made on which types of messages are to be received ("Subscribe"). Then, received messages are collected in a message queue, from where they can be read and evaluated.

Project structure

1. Create a TwinCAT project with a PLC and add Tc3_lotBase as library reference.
2. Create a program block and declare an instance of `FB lotMqttClient` [► 40] and two auxiliary variables to control the program sequence, if required.

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

3. Declare two variables (for topic and payload) for the MQTT message to be sent. In the sample a message is to be sent every second.

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

4. For each message receive operation a variable containing the topic to be received should be declared, plus two further variables indicating the topic and payload of the last received message. The received messages are to be collected in a queue for subsequent evaluation on a one-by-one basis. To this end, you should declare an instance of `FB lotMqttMessageQueue` [► 48] and an instance of `FB lotMqttMessage` [► 49].

```
(* 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. In the program part, the MQTT client must be triggered cyclically, in order to ensure that a connection to the broker is established and maintained and the message is received. Set the parameters of the desired connection and initialize the connection with the transfer parameter `bConnect := TRUE`.

In the sample the parameters are assigned once in the program code before the client call. Since this is usually only required once, the parameters can already be specified in the declaration part during instantiation of the MQTT client. Not all parameters have to be assigned. In the sample the broker is local. The IP address or the name can also be specified.

```
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. As soon as the connection to the broker is established, the client should subscribe to a particular topic. A message should be sent every second.

In the sample `sTopicPub = sTopicSub` applies, so that a loop-back occurs. In other applications the topics usually differ.

```
IF fbMqttClient.bConnected THEN
  IF NOT bSubscribed THEN
    bSubscribed := fbMqttClient.Subscribe(sTopic:=sTopicSub, eQoS:=TcIotMqttQos.AtMostOnceDelivery);
  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. The cyclic call of the MQTT client ensures that the messages are received. The client receives all messages with topics to which it has previously subscribed with the broker and places them in the message queue. Once messages are available, call the method `Dequeue()` to gain access to the message properties such as topic or payload via the message object `fbMessage`.

```
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
```

If message evaluation is implemented as described above, one received message is evaluated per cycle. If several messages were accumulated in the message queue, the evaluation is distributed over several cycles.

The sample can be modified for applications in which subscriptions to several topics exist. In this case MQTT messages with different topics are received. Message evaluation can be expanded as follows:

```
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), bSet
    etNullTermination:=FALSE);
    ELSEIF fbMessage.CompareTopic(sTopic:=sTopicSub2) THEN
      fbMessage.GetPayload(pPayload:=ADR(sPayloadRcv2), nPayloadSize:=SIZEOF(sPayloadRcv2), bS
    etNullTermination:=FALSE);
    END_IF
  END_IF
END_IF
```

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.0	IPC or CX (x86, x64, ARM)	Tc3_lotBase, Tc2_Uilities (>= v3.3.19.0)

6.2 lotMqttSampleUsingCallback

Sample for MQTT communication via a callback method

This sample illustrates the communication with an MQTT broker. Messages are sent (publish mode) and received. This is done in two steps. First, a general decision is made on which types of messages are to be received ("Subscribe"). Subsequently, new messages are received via a callback method during the cyclic call of the FB_lotMqttClient.Execute() method.

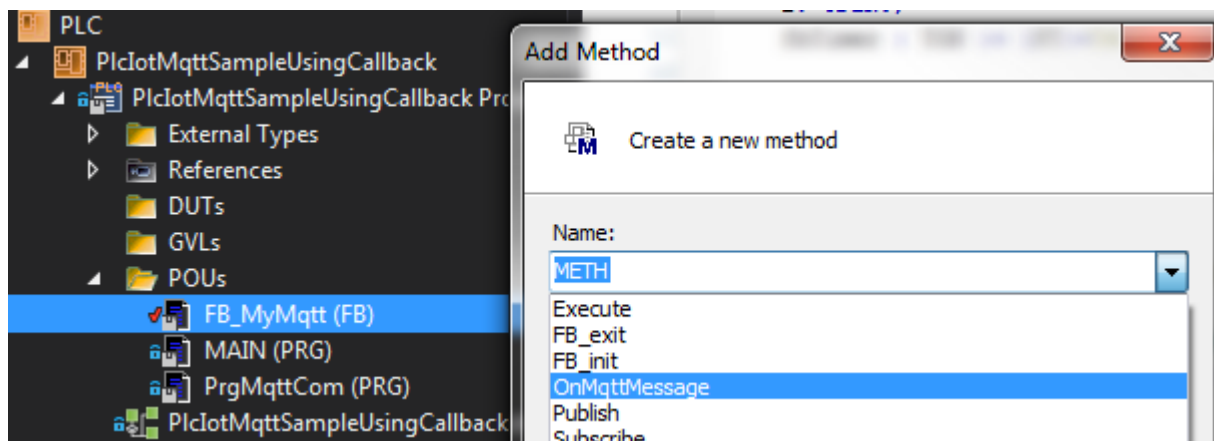
Project structure

1. Create a TwinCAT project with a PLC and add Tc3_lotBase as library reference.
2. The callback method, in which the received MQTT messages are provided, should be implemented by users themselves. The inheritance principle is used to ensure that the TwinCAT driver can call this method. First, create a function block and let the function block FB_lotMqttClient inherit it. Part of the MQTT communication can already be encapsulated in this function block. In the sample, received messages are evaluated here. It is therefore advisable to declare variables for topic and 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. Create the method OnMqttMessage() and overwrite the basic implementation.



4. The method with the implementation to be carried out by the user is not called in the application, but implicitly by the driver. This callback takes place during the cyclic triggering of the client and can take place either not at all, once or several times, depending on the number of messages received since the last trigger. This sample only implements a simple evaluation, as shown in the following code snippet.

```
{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. The other steps are similar to the sample [MQTT communication via a message queue \[► 135\]](#). Create a program block and declare an instance of the previously declared function block FB_MyMqtt and two auxiliary variables to control the program sequence, if required.

```

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

```

6. Declare two variables (for topic and payload) for the MQTT message to be sent. In the sample a message is to be sent every second.

```

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

```

7. To receive messages, declare a variable that contains the topic to be received.

```

bSubscribed : BOOL;
sTopicSub   : STRING(255) := 'MyTopic';

```

8. In the program part, the MQTT client must be triggered cyclically, in order to ensure that a connection to the broker is established and maintained and the message is received. Set the parameters of the desired connection and initialize the connection with the transfer parameter `bConnect := TRUE`. In the sample the parameters are assigned once in the program code before the client call. Since this is usually only required once, the parameters can already be specified in the declaration part during instantiation of the MQTT client. Not all parameters have to be assigned. In the sample the broker is local. The IP address or the name can also be specified.

```

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. As soon as the connection to the broker is established, the client should subscribe to a particular topic. A message should be sent every second.

In the sample `sTopicPub = sTopicSub` applies, so that a loop-back occurs. In other applications the topics usually differ.

```

IF fbMqttClient.bConnected THEN
  IF NOT bSubscribed THEN
    bSubscribed := fbMqttClient.Subscribe(sTopic:=sTopicSub, eQoS:=TcIotMqttQos.AtMostOnceDelivery);
  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

```

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.0	IPC or CX (x86, x64, ARM)	Tc3_lotBase, Tc2_Uilities (>= v3.3.19.0)

6.3 lotMqttSampleTlsPsk

Sample for MQTT communication via a secure TLS connection and PSK (PreSharedKey)

This sample illustrates the communication with an MQTT broker that requires authentication via TLS PSK. The sample is basically limited to establishing the connection and publishing of values.

Project structure

1. Create a TwinCAT project with a PLC and add Tc3_lotBase as library reference.
2. Create a program block and declare an instance of [FB_lotMqttClient](#) [► 40] and two auxiliary variables to control the program sequence, if required.

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

3. Declare two variables (for topic and payload) for the MQTT message to be sent. In the sample a message is to be sent every second.

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

4. In the program part the MQTT client must be triggered cyclically, in order to ensure that a connection to the broker is established and maintained. Set the parameters of the desired connection and initialize the connection with the transfer parameter `bConnect := TRUE`. In the sample the parameters are assigned once in the program code before the client call. Since this is usually only required once, the parameters can already be specified in the declaration part during instantiation of the MQTT client. Not all parameters have to be assigned. In the sample the broker is local. The IP address or the name can also be specified.

```
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. The structure element `aPskKey` receives the `PreSharedKey`, which is required for establishing a connection to the broker. Accordingly, this must be specified as an `ARRAY OF BYTE` with a length of 64. The actual length of the keys is then transferred to the structure element `nPskKeyLen`.
6. Once the connection to the broker is established, the client should send a message to a particular topic every second.

```
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

```

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.0	IPC or CX (x86, x64, ARM)	Tc3_lotBase, Tc2_Uilities (>= v3.3.19.0)

6.4 lotMqttSampleTlsCa

Sample for MQTT communication via a secured TLS connection and CA

This sample illustrates the communication with an MQTT broker that requires authentication via TLS and a client certificate. This sample is not available as a separate download, since it is essentially based on the existing samples [lotMqttSampleUsingQueue \[► 135\]](#) and in particular [lotMqttSampleAwsIoT \[► 140\]](#). The latter demonstrates the application of client certificates with TF6701 and can be used in the same way for all other MQTT brokers.

Parameters for establishing a connection

The following code snippet shows the parameters required for establishing a TLS connection to an MQTT broker via client certificate. The parameters are essentially static parameters. These can also be specified in the declaration part during instantiation of the MQTT client.

```

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

```

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.0	IPC or CX (x86, x64, ARM)	Tc3_lotBase, Tc2_Uilities (>= v3.3.19.0)

6.5 lotMqttSampleAwsIoT

Sample for MQTT communication with AWS IoT Core

This sample illustrates the communication with the AWS IoT Core message broker, which is part of the AWS IoT platform. The message broker requires authentication via a TLS client certificate. As a prerequisite for this, the corresponding certificate must have been created and be known and activated on the AWS IoT platform.

● Initial setup of AWS IoT Core

i Information on creating and registering client certificates and the initial setup of AWS IoT Core can be found in the [official AWS IoT Core documentation](#). The certificate created and activated there is used by the MQTT function blocks to establish a connection with the message broker. Ensure that you have linked a valid AWS IoT policy with the certificate you created. Further information can be found in the following articles, which are part of the AWS IoT Core documentation:

[AWS IoT Core Security and Identity](#)

[X.509 Certificates Authentication](#)

● Topic structure

i The topic structure of the AWS IoT Core message broker is essentially freely selectable, although some restrictions apply. Certain system topics cannot be used. Please refer to the AWS IoT Core documentation for more information. We also recommend the AWS documentation on [AWS IoT Core MQTT topic design](#).

● QoS and Retain

i AWS IoT Core currently does not support QoS 2 or Retain messages. To store persistent messages, additional services, such as AWS IoT Device Shadow or a database service, must be used.

● AWS IoT Core Service Limits

i When using AWS IoT Core, please also refer to the notes on [AWS Service Limits](#).

In this sample, messages are sent to and received from the AWS IoT Core message broker. Since this sample is essentially based on the sample [lotMqttSampleUsingQueue \[► 135\]](#), only the parts that are relevant for establishing a connection to AWS IoT Core are explained in this section.

Parameters for establishing a connection

The following code snippet shows the parameters required for establishing a connection to AWS IoT Core. The parameters are essentially static parameters. These can also be specified in the declaration part during instantiation of the MQTT client.

The following information is required for establishing a connection to AWS IoT Core.

Parameter	Description
fbMqttClient.stTLS.sCA	Root CA certificate, which is regarded as trusted by the AWS IoT broker. If certificates generated by the AWS IoT platform are used ("one-click certificate"), the corresponding root certificate of AWS IoT can be used. It can be downloaded from the AWS IoT website when certificates are generated.
fbMqttClient.stTLS.sCert	Certificate of the "thing", which is required as authentication for establishing a connection to AWS IoT. If the certificates generated by the AWS IoT platform are used ("one-click certificate"), this certificate can be downloaded from the AWS IoT website, once it has been generated.
fbMqttClient.stTLS.sKeyFile	Private key of the "thing", which is required for securing the communication link. If the certificates generated by the AWS IoT platform are used ("one-click certificate"), the private key can be downloaded from the AWS IoT website, once it has been generated.
fbMqttClient.sHostname	FQDN of the AWS IoT broker instance
fbMqttClient.nHostPort	Since a connection to the AWS IoT broker can only be established via TLS, the default MQTT TLS port must be used here (8883).
fbMqttClient.sClientId	The MQTT client ID can be the same as the name of the "thing".
Exponential backoff	The use of an exponential backoff algorithm, as shown in the code snippet, is optional but recommended. See below.

```
IF bSetParameter THEN
  bSetParameter := FALSE;
  fbMqttClient.stTLS.sCA := 'c:\TwinCAT\3.1\Config\Certificates\root.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

A feature referred to as "exponential backoff" can be used to avoid burdening the message broker with unnecessary connection requests in case of a connection error. In the event of a TLS connection error involving the message broker, the reconnect rate is adjusted multiplicatively. This function can be activated using the [ActivateExponentialBackoff\(\)](#) [► 46] method. The parameters of the method specify the minimum and maximum time for the algorithm. The minimum time describes the initial delay value for the new connection attempt. The maximum time describes the highest delay value. The delay values are doubled until the maximum value is reached. Once a connection has been established, the backoff rate is reset to the original value. The [DeactivateExponentialBackoff\(\)](#) [► 46] method can be used to deactivate this function programmatically.

Device Shadow Service

The AWS IoT Device Shadow Service enables persistent storage of status information of a connected device. A separate shadow is managed for each device. The shadow of a device can be read and updated via MQTT. Certain system topics of the AWS IoT Core must be used for this purpose. For example, the following topic enables updating the shadow.

```
sTopicShadowUpdate : STRING(255) := '$$aws/things/CX-12345/shadow/update';
```

The message sent to this topic describes the new shadow of the device. It is specified in JSON notation and corresponds to a particular format, e.g.:

```
{
  "state": {
    "reported": {
      "Vendor": "Beckhoff Automation",
      "CpuTemperature": 42,
      "OperatingSystem": "Windows 10"
    }
  }
}
```

The PLC library Tc3_JsonXml can be used to create and adapt this format to suit your application.

In the PLC code provided here, the device shadow is updated once at the start of the sample and then on request (depending on the variable bUpdateShadow).

For more information about the AWS IoT Device Shadow Service and its topics and data formats see the [AWS IoT Device Shadow Service documentation](#).

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.0	IPC or CX (x86, x64, ARM)	Tc3_lotBase, Tc2_Uilities (>= v3.3.19.0)

6.6 lotMqttSampleAzureIoT Hub

Sample for MQTT communication with the Microsoft Azure IoT Hub

This sample illustrates the communication with the Microsoft Azure IoT Hub, which is part of the Microsoft Azure cloud. The message broker can be reached via MQTT and requires authentication via an SAS token, which can be generated via the Azure IoT Hub platform, e.g. using the Azure IoT Explorer.

Initial setup of Azure IoT Hub

For information on the initial setup of the Microsoft Azure IoT Hub and corresponding access data for devices to be connected, see the [official Microsoft Azure IoT Hub documentation](#). We also recommend the Microsoft documentation article on [using MQTT with the Azure IoT Hub](#).

● Topic structure

i The topic structure for sending and receiving messages is predefined by the Microsoft Azure IoT Hub.

● SAS token

i The use of an SAS token is a basic requirement for connecting to the Azure IoT Hub via MQTT. This token can be generated with the Azure IoT Explorer, for example.

● QoS and Retain

i The Azure IoT Hub does not support QoS 2 and Retain messages.

In this sample messages are sent to the Azure IoT Hub and received from it. Since this sample is essentially based on the sample [lotMqttSampleUsingQueue](#) [► 135], only the parts that are relevant for establishing a connection to the IoT Hub are explained in this section.

Parameters for establishing a connection

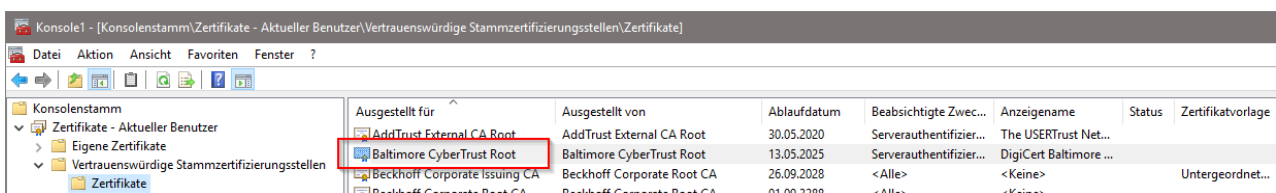
The following code snippet shows the parameters required for establishing a connection to the Azure IoT Hub. The parameters are essentially static parameters. These can also be specified in the declaration part during instantiation of the MQTT client.

```
IF bSetParameter THEN
  bSetParameter := FALSE;
  // fbMqttClient.stTLS.sCA := 'c:\TwinCAT\3.1\Config\Certificates\azure.crt';
  fbMqttClient.stTLS.sAzureSas := 'HostName=xxx.azure-
devices.net;DeviceId=xxx;SharedAccessSignature=SharedAccessSignature sr=xxx.azure-
devices.net%2fdevices%2fXXX&sig=121b5gJZxujK5pV%2bsFIFc2nddtpuhRuY7Tjfn8kJbtA%3d&se=1490275463';
END_IF
```

The structure element `fbMqttClient.stTLS.sAzureSas` enables transfer of the SAS token, which can be generated with the Azure IoT Explorer. The use of the structure element `fbMqttClient.stTLS.sCA` is optional. By default the driver uses the path specified above in the background. A root CA that is accepted by the Azure IoT Hub should be stored under this path. The topics for publish and subscribe are specified by the Azure IoT Hub and cannot be changed.

● CA certificate

i When establishing a connection to the Microsoft Azure IoT Hub via MQTT, the specification of a CA certificate is mandatory. The Baltimore Cyber Trust Root CA can be used as CA certificate. The corresponding public key can be extracted from the Microsoft Windows certificate console (**Start > Run > mmc.exe**, then add the **SnapIn** certificate). The Baltimore CA can then be found under the heading **Trusted Root Certification Authorities**.

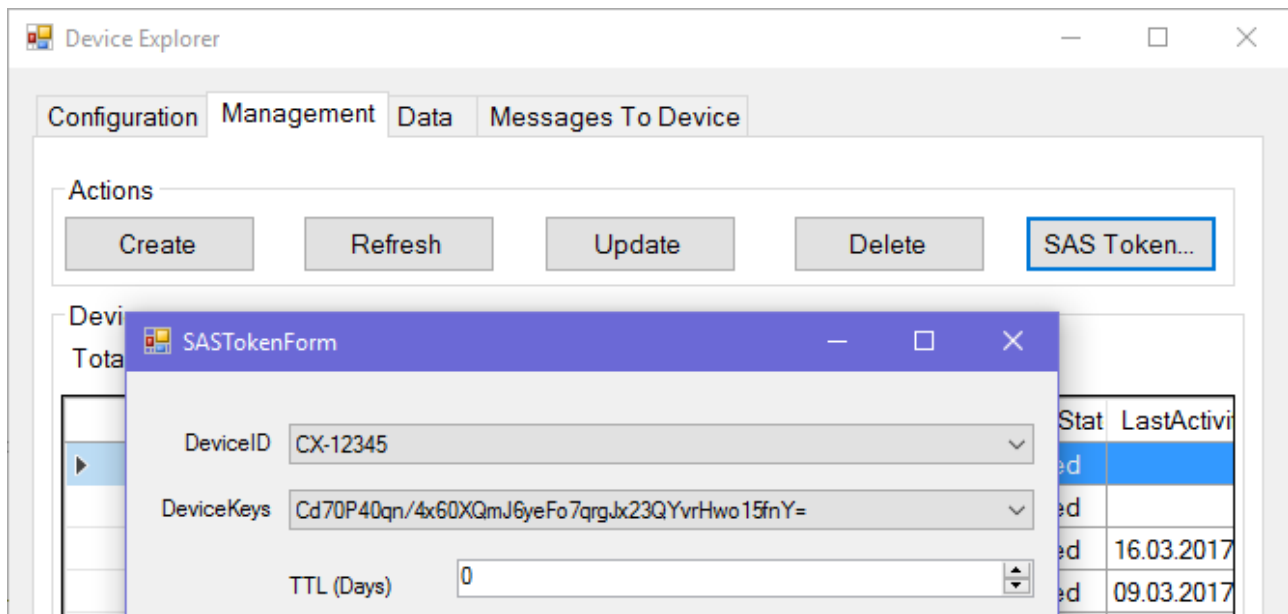


Publish

When data are published to the IoT Hub, the topic must be specified in the following form:

devices / deviceId / messages / events / readpipe

The DeviceId corresponds to the DeviceId of the registered device, as specified in the Azure IoT Explorer, for example.



Subscribe

For subscribing to data from the IoT Hub, the topic must be specified in the following form:

devices / deviceId / messages / devicebound / #

The DeviceId corresponds to the DeviceId of the registered device, as specified in the Azure IoT Explorer, for example.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.0	IPC or CX (x86, x64, ARM)	Tc3_lotBase, Tc2_Uilities (>= v3.3.19.0)

6.7 lotMqttSampleBoschIoT

Sample for MQTT communication with the Bosch IoT Hub

This sample shows the communication with the Bosch IoT Hub, which is part of the Bosch IoT Suite. This message broker requires the use of TLS and user name/password authentication.

Initial setup of the Bosch IoT Hub

Information on the initial setup of the Bosch IoT Hub can be found in the [official Bosch IoT Suite documentation](#).

Topic structure

The topic structure for sending and receiving messages is predefined by the message broker of the Bosch IoT Suite.

In this sample messages are sent to the Bosch IoT Hub and received from it. Since this sample is essentially based on the sample [lotMqttSampleUsingQueue](#) [► 135], only the parts that are relevant for establishing a connection to the Bosch IoT Hub are explained in this section.

Parameters for establishing a connection

The following code snippet shows the parameters required for establishing a connection to the Bosch IoT Hub. The parameters are essentially static parameters. These can also be specified in the declaration part during instantiation of the MQTT client.

```

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

```

The parameters required for authentication can be generated on the Bosch IoT platform.

6.8 lotMqttSampleGoogleIoT

Sample for MQTT communication with Google IoT Core

This sample illustrates the communication to the Google IoT Core message broker, which is part of the Google Cloud. The message broker requires TLS and authentication via a JSON Web Token (JWT).

● Initial setup of Google IoT Core

i Information on creating and registering devices and the initial setup of Google IoT Core can be found in the [official Google IoT Core documentation](#).

● JSON Web Token (JWT)

i The JSON Web Token (JWT) feature is used to connect to Google IoT Core. Tokens can be generated via the function block FB_JwtEncode from the library Tc3_JsonXml.

● QoS and Retain

i Google IoT Core currently does not support QoS 2 or Retain messages.

In this sample, messages are sent to and received from the Google IoT Core message broker. Since this sample is essentially based on the sample [lotMqttSampleUsingQueue](#) [► 135], only the parts that are relevant for establishing a connection to Google IoT Core are explained in this section.

Parameters for establishing a connection

The following code snippet shows the parameters required for establishing a connection to Google IoT Core. The parameters are essentially static parameters. These can also be specified in the declaration part during instantiation of the MQTT client.

```

IF bSetParameter THEN
  bSetParameter := FALSE;
  fbMqttClient.sHostName:= 'mqtt.googleapis.com';
  fbMqttClient.nHostPort:= 8883;
  fbMqttClient.sClientId:= 'projects/%projectName%/locations/europe-west1/registries/%registryName%/
devices/%deviceName%';
  fbMqttClient.sUserName:= 'unused';
  fbMqttClient.sUserPassword:= sJwt;
  fbMqttClient.stTLS.sCA := 'c:\TwinCAT\3.1\Config\Certificates\Google\roots.pem';
END_IF

```

The MQTT client ID is derived from the above string and contains the project name, registry name and device name. These three parameters are generated when the Google IoT Core service is first set up and a device is created there. The user name is not used. According to the Google documentation, "unused" can be used. The password is a [JSON Web Token \(JWT\)](#) [► 28], which can be generated from the Tc3_JsonXml library via the function block FB_JwtEncode and signed with the private key of the device certificate. The CA certificate can be obtained from the Google IoT Core website.

Renewal of the JSON web token

As mentioned earlier in this article, a JWT is used for authentication with the Google IoT Core Broker. This JWT has an expiration date after which it can no longer be used for authentication and is therefore obsolete. This is the case after a maximum of 24 hours.

To renew the JWT, the existing connection to the message broker must be terminated, the token exchanged and the connection re-established. The following code snippet shows this workflow (not included in the sample above).

```
//Cycle Time is set to 10 ms
refreshCounter:=refreshCounter+1;
//Refreshing every 5 minutes
IF refreshCounter>30000 THEN
    refreshCounter:=0;
    bConnect:=FALSE;
    fbMqttClient.Execute(bConnect);
ELSE
    bConnect:=TRUE;
    fbMqttClient.Execute(bConnect);
END_IF
```

6.9 lotMqttSampleIbmWatsonIoT

Sample for MQTT communication with IBM Watson IoT

This sample shows the communication with IBM Watson IoT.



Topic structure

The topic structure for sending and receiving messages is predefined by the IBM Watson IoT message broker.

Messages are sent to and received from IBM Watson IoT. Since this sample is essentially based on the sample [lotMqttSampleUsingQueue](#) [► 135], only the parts that are relevant for establishing a connection are explained in this section.

Parameters for establishing a connection

The following code snippet shows the parameters required for establishing a connection to IBM Watson IoT. The parameters are essentially static parameters. These can also be specified in the declaration part during instantiation of the MQTT client.

```
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
```

The topics for publish and subscribe are specified by IBM Watson IoT and cannot be changed. The “orgID” placeholder must be replaced with the organization ID of the IBM Watson account. The “deviceId” placeholder is replaced with the ID of the device, as created in IBM Watson.

Publish

When data are published to IBM Watson IoT, the topic must be specified in the following form:

iot-2 / evt / eventId / fmt / json

The event ID corresponds to the event ID as configured and expected by IBM Watson IoT. If an IBM Watson dashboard is used, the event ID is generated dynamically and can be linked to a chart on IBM Watson IoT.

Subscribe

When subscribing to IBM Watson IoT commands, the topic must be specified in the following form:

iot-2 / cmd / cmdId / fmt / json

The Cmd ID corresponds to the ID of the command sent by IBM Watson IoT to the device.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.0	IPC or CX (x86, x64, ARM)	Tc3_IotBase, Tc2_Uilities (>= v3.3.19.0)

6.10 lotMqttSampleMathworksThingspeak

Communication with the MathWorks ThingSpeak Cloud is illustrated in this sample. Since this sample is essentially based on the sample [lotMqttSampleUsingQueue \[► 135\]](#), only the parts that are relevant for establishing a connection to the ThingSpeak Cloud are explained in this section.

Sample code can be downloaded as archive here: https://infosys.beckhoff.com/content/1033/tf6701_tc3_iot_communication_mqtt/Resources/11990435339/.zip.

MQTT device configuration

To transfer MQTT data to ThingSpeak, you must first register your MQTT device. Log in to ThingSpeak with your MathWorks account and select **Devices > MQTT** from the top menu. Select **Add new Device**. Name your device and authorize *publish* and *subscribe* on the appropriate ThingSpeak channels. Save your credentials.

Parameters for establishing a connection

The following code snippet illustrates the parameters that are necessary for establishing a connection with the MathWorks ThingSpeak Cloud. The parameters are essentially static parameters. These can also be specified in the declaration part during instantiation of the MQTT client. The data to be used in the following <Client ID>, <Username> and <Password> will be provided to you when you configure an MQTT device on ThingSpeak.

```
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.UserPassword:= '<password as provided by MQTT Device on ThingSpeak>';
END_IF
```

ThingSpeak also supports protection of the communication connection via TLS. The CA certificate must be downloaded from ThingSpeak and referenced in the function block via the TLS structure. For more information about the ThingSpeak CA certificate we recommend referring to the Mathworks ThingSpeak documentation.

```
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.UserPassword:= '<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

When publishing data to the MathWorks ThingSpeak Cloud, the topic must be specified in the following form:

channels / <channelID> / publish

<channelID> corresponds here to the ID of the channel that was designated and configured for data reception in the MathWorks ThingSpeak portal.

Your device must be authorized for *publish* on the specified channel. You can change the authorization via **Devices > MQTT** in the top menu if you are logged in to the ThingSpeak website.

Subscribe

When subscribing to data, the topic must be specified in the following form:

channels / <channelID> / subscribe / fields / <fieldKey>

<channelID> corresponds here to the ID of the channel that was designated and configured for data reception in the MathWorks ThingSpeak portal.

<fieldKey> corresponds to the name of the field from which a value is to be received.

Your device must be authorized for a *subscribe* on the specified channel. You can change the authorization via **Devices > MQTT** in the top menu when you are logged in to the ThingSpeak website

Data format

The MathWorks ThingSpeak Cloud uses its own string-based data format as payload. This data format is generated in the sample code referenced above in the `F_Mqtt_ThingSpeak_CreatePayloadStr()` function and can be used directly.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.0	IPC or CX (x86, x64, ARM)	Tc3_lotBase, Tc2_Uilities (>= v3.3.19.0)

6.11 lotMqttSampleAzureIoTDeviceTwin

Sample for MQTT communication with the Device Twin of the Azure IoT Hub

This sample illustrates the communication with the Device Twin. The Device Twin is accessible via the Azure IoT Hub, which is part of the Microsoft Azure Cloud. The message broker can be reached via MQTT and requires authentication via an SAS token, which can be generated via the Azure IoT Hub platform, e.g. using the Azure IoT Explorer.

The following sample connects to the Azure IoT Hub, just like the sample `lotMqttSampleAzureIoTDeviceTwin` [► 142]. All relevant information for establishing the connection and further configuration options can be found in this article.

The official MQTT documentation from Microsoft can be downloaded from here:

<https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-mqtt-support>

6.12 JsonXmlSamples

6.12.1 Tc3JsonXmlSampleXmlDomWriter

This sample illustrates how an XML document can be created programmatically based on DOM. The function block `FB_XmlDomParser` is used as a basis.

Download: https://infosys.beckhoff.com/content/1033/tf6701_tc3_iot_communication_mqtt/Resources/5529228299/.zip

Declaration range

```
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
```

Implementation range

The implementation section shows various options for creating an XML document.

```
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
```

6.12.2 Tc3JsonXmlSampleXmlDomReader

This sample illustrates how an XML document can be processed programmatically based on DOM. The function block FB_XmlDomParser is used as a basis.

Download: https://infosys.beckhoff.com/content/1033/tf6701_tc3_iot_communication_mqtt/Resources/5529225227/.zip

Declaration range

```
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
```

Implementation range

The implementation section shows various options for parsing an XML document.

```
(* 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);
```

6.12.3 Tc3JsonXmlSampleJsonDomReader

This sample illustrates how a JSON message can be run through programmatically on the basis of DOM. The function block FB_JsonDomParser is used as the basis.

Download: https://infosys.beckhoff.com/content/1033/tf6701_tc3_iot_communication_mqtt/Resources/3916597387.zip

Declaration range

```
PROGRAM MAIN
VAR
    fbJson      : FB_JsonDomParser;
    jsonDoc     : SJJsonValue;
    jsonProp    : SJJsonValue;
    jsonValue   : SJJsonValue;
    bHasMember  : BOOL;
    sMessage    : STRING(255) := '{"serialNumber":"G030PT028191AC4R","batteryVoltage":"1547mV","clickType":"SINGLE"}';
    stReceivedData : ST_ReceivedData;
END_VAR
```

Implementation range

The JSON message is loaded into the DOM tree using the ParseDocument() method. You can subsequently check whether it contains a certain property using the HasMember() method. The FindMember() method selects the property. The GetString() method extracts its value.

```
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
```

The use of the method HasMember() is not absolutely necessary, since the method FindMember() already returns 0 if a property was not found. The code shown above can also be implemented as follows:

```
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
```

Nested JSON objects

The approach is similar with nested JSON objects. Since the entire document is located in the DOM, it is simple to navigate. Let's take a JSON object that looks like this:

```
sMessage : STRING(255) := '{"Values":{"serial":"G030PT028191AC4R"}}';
```

The property we are looking for is located in the sub-object "Values". The following code shows how to extract the property.

```
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
```

6.12.4 Tc3JsonXmlSampleJsonSaxWriter

Sample of the creation of JSON documents via SAX Writer

This sample illustrates how a JSON message can be created over the DAX mechanism. The function block FB_JsonSaxWriter is used as the basis.

Download: https://infosys.beckhoff.com/content/1033/tf6701_tc3_iot_communication_mqtt/Resources/3664753419/.zip

Declaration range

```
PROGRAM MAIN
VAR
  dtTimestamp : DATE_AND_TIME := DT#2017-04-04-12:42:42;
  fbJson      : FB_JsonSaxWriter;
  sJsonDoc    : STRING(255);
END_VAR
```

Implementation range

The SAX mechanism runs sequentially through the JSON document to be created, i.e. the corresponding elements are run through and created one after the other.

```
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();
```

Resulting JSON message

```
{
  "Timestamp": "2017-04-04T12:42:42",
  "Values": {
    "Sensor1": 42.42,
    "Sensor2": 42,
    "Sensor3": true
  }
}
```

6.12.5 Tc3JsonXmlSampleJsonSaxReader

Sample of the parsing of JSON documents via SAX Reader

This sample illustrates how a JSON message can be run through programmatically. The function block FB_JsonSaxReader is used as the basis.

Download: https://infosys.beckhoff.com/content/1033/tf6701_tc3_iot_communication_mqtt/Resources/3664750475.zip

Declaration range

```
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
```

Implementation range

Through the calling of the Parse() method, the transfer of the JSON message as a STRING and the interface pointer to a function block instance that implements the interface ItcJsonSaxHandler, the SAX Reader is activated and the corresponding callback methods are run through.

```
fbJson.Parse(sJson := sJsonDoc, ipHdl := pJsonParse);
```

Callback methods

The callback methods are called on the instance of the function block that implements the interface ItcJsonSaxHandler. Each callback method represents a "found" element in the JSON message. For example, the callback method OnStartObject() is called as soon as an opening curly bracket has been detected. According to the example JSON message mentioned above, therefore, the following callback methods are run through in this order:

1. OnStartObject(), due to the first opening curly bracket
2. OnKey(), due to the property "Values"
3. OnStartObject(), due to the second opening curly bracket
4. OnKey(), due to the property "Timestamp"
5. OnString(), due to the value of the property "Timestamp"
6. OnKey(), due to the property "Sensor1"
7. OnLreal(), due to the value of the property "Sensor1"
8. OnKey(), due to the property "Sensor2"
9. OnUdint(), due to the value of the property "Sensor2"
10. OnEndObject(), due to the first closing curly bracket
11. OnEndObject(), due to the second closing curly bracket

Within the callback methods the current state is defined and saved via an instance of the enum E_JsonStates. This can also be used to determine whether the JSON message is valid. For example, if the callback method OnLreal() is called and the state is not the expected State 70 (JSON_STATE_ONLREAL), the return value S_FALSE can be returned to the method. The SAX Reader then automatically cancels the further processing.

6.12.6 Tc3JsonXmlSampleJsonDataType

Sample of the automatic conversion of structures into a JSON message

This sample illustrates how a data structure can be converted into a JSON message (and vice versa). In the conversion the layout of a structure is converted one-to-one into a corresponding JSON equivalent. Additional metadata can be created via PLC attributes on the member variables of the structure.

Download: https://infosys.beckhoff.com/content/1033/tf6701_tc3_iot_communication_mqtt/Resources/3664376331.zip

Layout of the data structure to be converted

```

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

```

Declaration range

```

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

```

Implementation range

Two ways of generating the JSON message are shown, starting with the instance fbJson of the function block FB_JsonSaxWriter. The GetDocument() method can be used with a JSON message with no more than 255 characters. However, the CopyDocument() method must be used with larger JSON messages.

```

fbJson.ResetDocument();
fbJson.StartObject();
fbJson.AddKeyDateTime('Timestamp', dtTimestamp);
fbJsonDataType.AddJsonKeyValueFromSymbol(fbJson, 'Values', 'ST_Values', SIZEOF(stValues), ADR(stValues));
fbJsonDataType.AddJsonKeyPropertiesFromSymbol(fbJson, 'MetaData', 'ST_Values', 'Unit|DisplayName');
fbJson.EndObject();
sJsonDoc := fbJson.GetDocument();
fbJson.CopyDocument(sJsonDoc2, SIZEOF(sJsonDoc2));

```

Resulting JSON message

```

{
  "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

As an alternative, the method `AddJsonValueFromSymbol()` can also be used to generate a JSON format directly from a data structure.

```
fbJson.ResetDocument();
fbJsonDataType.AddJsonValueFromSymbol(fbJson, 'ST_Values', sizeof(stValues), ADR(stValues));
sJsonDoc := fbJson.GetDocument();
fbJson.CopyDocument(sJsonDoc2, sizeof(sJsonDoc2));
```

The resulting JSON object looks like this:

```
{
  "Sensor1": 0.0,
  "Sensor2": 0,
  "Sensor3": 0
}
```

Conversion of a JSON message back to a data structure

The above samples show how a JSON object can be generated from a data structure in a simple manner. There is also a corresponding method in the `Tc3_JsonXml` library for the reverse process, i.e. the extraction of values from a (received) JSON object back into a data structure. This application is made possible by calling the method `SetSymbolFromJson()`.

```
fbJsonDataType.SetSymbolFromJson(someJson, 'ST_Values', sizeof(stValuesReceive),
ADR(stValuesReceive));
```

The string variable `sJsonDoc2` contains the JSON object, which is transferred into the structure instance `stValuesReceive` by calling the method.



Target data structure

The target data structure must match the structure of the JSON document. Otherwise `SetSymbolFromJson()` returns `FALSE`.

7 Appendix

7.1 Error Codes

In the event of an error, the function block FB `lotMqttClient` [► 40] sets the output `bError` and indicates the error with `hrErrorCode`. All errors are listed in section "ADS Return Codes [► 156]".

In addition, the output `eConnectionState` indicates the state of the connection between the client and the MQTT broker at all times. The enumeration offers the following possible states:

```
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
```

7.2 ADS Return Codes

Grouping of error codes: `0x000` [► 156]..., `0x500` [► 157]..., `0x700` [► 158]..., `0x1000` [► 160]...

Global error codes

Hex	Dec	HRESULT	Name	Description
0x0	0	0x9811 0000	ERR_NOERROR	No error.
0x1	1	0x9811 0001	ERR_INTERNAL	Internal error.
0x2	2	0x9811 0002	ERR_NORTIME	No real-time.
0x3	3	0x9811 0003	ERR_ALLOCLOCKEDMEM	Allocation locked – memory error.
0x4	4	0x9811 0004	ERR_INSERTMAILBOX	Mailbox full – the ADS message could not be sent. Reducing the number of ADS messages per cycle will help.
0x5	5	0x9811 0005	ERR_WRONGRECEIVEHMSG	Wrong HMSG.
0x6	6	0x9811 0006	ERR_TARGETPORTNOTFOUND	Target port not found – ADS server is not started or is not reachable.
0x7	7	0x9811 0007	ERR_TARGETMACHINENOTFOUND	Target computer not found – AMS route was not found.
0x8	8	0x9811 0008	ERR_UNKNOWNCMDID	Unknown command ID.
0x9	9	0x9811 0009	ERR_BADTASKID	Invalid task ID.
0xA	10	0x9811 000A	ERR_NOIO	No IO.
0xB	11	0x9811 000B	ERR_UNKNOWNAMSCMD	Unknown AMS command.
0xC	12	0x9811 000C	ERR_WIN32ERROR	Win32 error.
0xD	13	0x9811 000D	ERR_PORTNOTCONNECTED	Port not connected.
0xE	14	0x9811 000E	ERR_INVALIDAMSLENGTH	Invalid AMS length.
0xF	15	0x9811 000F	ERR_INVALIDAMSNETID	Invalid AMS Net ID.
0x10	16	0x9811 0010	ERR_LOWINSTLEVEL	Installation level is too low –TwinCAT 2 license error.
0x11	17	0x9811 0011	ERR_NODEBUGINTAVAILABLE	No debugging available.
0x12	18	0x9811 0012	ERR_PORTDISABLED	Port disabled – TwinCAT system service not started.
0x13	19	0x9811 0013	ERR_PORTALREADYCONNECTED	Port already connected.
0x14	20	0x9811 0014	ERR_AMSSYNC_W32ERROR	AMS Sync Win32 error.
0x15	21	0x9811 0015	ERR_AMSSYNC_TIMEOUT	AMS Sync Timeout.
0x16	22	0x9811 0016	ERR_AMSSYNC_AMSERROR	AMS Sync error.
0x17	23	0x9811 0017	ERR_AMSSYNC_NOINDEXINMAP	No index map for AMS Sync available.
0x18	24	0x9811 0018	ERR_INVALIDAMSPORT	Invalid AMS port.
0x19	25	0x9811 0019	ERR_NOMEMORY	No memory.
0x1A	26	0x9811 001A	ERR_TCPSEND	TCP send error.
0x1B	27	0x9811 001B	ERR_HOSTUNREACHABLE	Host unreachable.
0x1C	28	0x9811 001C	ERR_INVALIDAMSFRAGMENT	Invalid AMS fragment.
0x1D	29	0x9811 001D	ERR_TLSEND	TLS send error – secure ADS connection failed.
0x1E	30	0x9811 001E	ERR_ACCESSDENIED	Access denied – secure ADS access denied.

Router error codes

Hex	Dec	HRESULT	Name	Description
0x500	1280	0x9811 0500	ROUTERERR_NOLOCKEDMEMORY	Locked memory cannot be allocated.
0x501	1281	0x9811 0501	ROUTERERR_RESIZEMEMORY	The router memory size could not be changed.
0x502	1282	0x9811 0502	ROUTERERR_MAILBOXFULL	The mailbox has reached the maximum number of possible messages.
0x503	1283	0x9811 0503	ROUTERERR_DEBUGBOXFULL	The Debug mailbox has reached the maximum number of possible messages.
0x504	1284	0x9811 0504	ROUTERERR_UNKNOWNPORTTYPE	The port type is unknown.
0x505	1285	0x9811 0505	ROUTERERR_NOTINITIALIZED	The router is not initialized.
0x506	1286	0x9811 0506	ROUTERERR_PORTALREADYINUSE	The port number is already assigned.
0x507	1287	0x9811 0507	ROUTERERR_NOTREGISTERED	The port is not registered.
0x508	1288	0x9811 0508	ROUTERERR_NOMOREQUEUES	The maximum number of ports has been reached.
0x509	1289	0x9811 0509	ROUTERERR_INVALIDPORT	The port is invalid.
0x50A	1290	0x9811 050A	ROUTERERR_NOTACTIVATED	The router is not active.
0x50B	1291	0x9811 050B	ROUTERERR_FRAGMENTBOXFULL	The mailbox has reached the maximum number for fragmented messages.
0x50C	1292	0x9811 050C	ROUTERERR_FRAGMENTTIMEOUT	A fragment timeout has occurred.
0x50D	1293	0x9811 050D	ROUTERERR_TOBEREMOVED	The port is removed.

General ADS error codes

Hex	Dec	HRESULT	Name	Description
0x700	1792	0x9811 0700	ADSERR_DEVICE_ERROR	General device error.
0x701	1793	0x9811 0701	ADSERR_DEVICE_SRVNOTSUPP	Service is not supported by the server.
0x702	1794	0x9811 0702	ADSERR_DEVICE_INVALIDGRP	Invalid index group.
0x703	1795	0x9811 0703	ADSERR_DEVICE_INVALIDOFFSET	Invalid index offset.
0x704	1796	0x9811 0704	ADSERR_DEVICE_INVALIDACCESS	Reading or writing not permitted.
0x705	1797	0x9811 0705	ADSERR_DEVICE_INVALIDSIZE	Parameter size not correct.
0x706	1798	0x9811 0706	ADSERR_DEVICE_INVALIDDATA	Invalid data values.
0x707	1799	0x9811 0707	ADSERR_DEVICE_NOTREADY	Device is not ready to operate.
0x708	1800	0x9811 0708	ADSERR_DEVICE_BUSY	Device is busy.
0x709	1801	0x9811 0709	ADSERR_DEVICE_INVALIDCONTEXT	Invalid operating system context. This can result from use of ADS function blocks in different tasks. It may be possible to resolve this through Multi-task data access synchronization in the PLC.
0x70A	1802	0x9811 070A	ADSERR_DEVICE_NOMEMORY	Insufficient memory.
0x70B	1803	0x9811 070B	ADSERR_DEVICE_INVALIDPARAM	Invalid parameter values.
0x70C	1804	0x9811 070C	ADSERR_DEVICE_NOTFOUND	Not found (files, ...).
0x70D	1805	0x9811 070D	ADSERR_DEVICE_SYNTAX	Syntax error in file or command.
0x70E	1806	0x9811 070E	ADSERR_DEVICE_INCOMPATIBLE	Objects do not match.
0x70F	1807	0x9811 070F	ADSERR_DEVICE_EXISTS	Object already exists.
0x710	1808	0x9811 0710	ADSERR_DEVICE_SYMBOLNOTFOUND	Symbol not found.
0x711	1809	0x9811 0711	ADSERR_DEVICE_SYMBOLVERSIONINVALID	Invalid symbol version. This can occur due to an on-line change. Create a new handle.
0x712	1810	0x9811 0712	ADSERR_DEVICE_INVALIDSTATE	Device (server) is in invalid state.
0x713	1811	0x9811 0713	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransMode not supported.
0x714	1812	0x9811 0714	ADSERR_DEVICE_NOTIFYHNDINVALID	Notification handle is invalid.
0x715	1813	0x9811 0715	ADSERR_DEVICE_CLIENTUNKNOWN	Notification client not registered.
0x716	1814	0x9811 0716	ADSERR_DEVICE_NOMOREHDL	No further handle available.
0x717	1815	0x9811 0717	ADSERR_DEVICE_INVALIDWATCHSIZE	Notification size too large.
0x718	1816	0x9811 0718	ADSERR_DEVICE_NOTINIT	Device not initialized.
0x719	1817	0x9811 0719	ADSERR_DEVICE_TIMEOUT	Device has a timeout.
0x71A	1818	0x9811 071A	ADSERR_DEVICE_NOINTERFACE	Interface query failed.
0x71B	1819	0x9811 071B	ADSERR_DEVICE_INVALIDINTERFACE	Wrong interface requested.
0x71C	1820	0x9811 071C	ADSERR_DEVICE_INVALIDCLSID	Class ID is invalid.
0x71D	1821	0x9811 071D	ADSERR_DEVICE_INVALIDOBJID	Object ID is invalid.
0x71E	1822	0x9811 071E	ADSERR_DEVICE_PENDING	Request pending.
0x71F	1823	0x9811 071F	ADSERR_DEVICE_ABORTED	Request is aborted.
0x720	1824	0x9811 0720	ADSERR_DEVICE_WARNING	Signal warning.
0x721	1825	0x9811 0721	ADSERR_DEVICE_INVALIDARRAYIDX	Invalid array index.
0x722	1826	0x9811 0722	ADSERR_DEVICE_SYMBOLNOTACTIVE	Symbol not active.
0x723	1827	0x9811 0723	ADSERR_DEVICE_ACCESSDENIED	Access denied.
0x724	1828	0x9811 0724	ADSERR_DEVICE_LICENSENOTFOUND	Missing license.
0x725	1829	0x9811 0725	ADSERR_DEVICE_LICENSEEXPIRED	License expired.
0x726	1830	0x9811 0726	ADSERR_DEVICE_LICENSEEXCEEDED	License exceeded.
0x727	1831	0x9811 0727	ADSERR_DEVICE_LICENSEINVALID	Invalid license.
0x728	1832	0x9811 0728	ADSERR_DEVICE_LICENSESYSTEMID	License problem: System ID is invalid.
0x729	1833	0x9811 0729	ADSERR_DEVICE_LICENSENOTIMELIMIT	License not limited in time.
0x72A	1834	0x9811 072A	ADSERR_DEVICE_LICENSEFUTUREISSUE	License problem: Time in the future.
0x72B	1835	0x9811 072B	ADSERR_DEVICE_LICENSESETIMETO LONG	License period too long.
0x72C	1836	0x9811 072C	ADSERR_DEVICE_EXCEPTION	Exception at system startup.
0x72D	1837	0x9811 072D	ADSERR_DEVICE_LICENSEDUPLICATED	License file read twice.
0x72E	1838	0x9811 072E	ADSERR_DEVICE_SIGNATUREINVALID	Invalid signature.
0x72F	1839	0x9811 072F	ADSERR_DEVICE_CERTIFICATEINVALID	Invalid certificate.
0x730	1840	0x9811 0730	ADSERR_DEVICE_LICENSEOEMNOT-FOUND	Public key not known from OEM.
0x731	1841	0x9811 0731	ADSERR_DEVICE_LICENSERESTRICTED	License not valid for this system ID.
0x732	1842	0x9811 0732	ADSERR_DEVICE_LICENSEDEMODENIED	Demo license prohibited.
0x733	1843	0x9811 0733	ADSERR_DEVICE_INVALIDFNCID	Invalid function ID.
0x734	1844	0x9811 0734	ADSERR_DEVICE_OUTOFRANGE	Outside the valid range.
0x735	1845	0x9811 0735	ADSERR_DEVICE_INVALIDALIGNMENT	Invalid alignment.

Hex	Dec	HRESULT	Name	Description
0x736	1846	0x9811 0736	ADSERR_DEVICE_LICENSEPLATFORM	Invalid platform level.
0x737	1847	0x9811 0737	ADSERR_DEVICE_FORWARD_PL	Context – forward to passive level.
0x738	1848	0x9811 0738	ADSERR_DEVICE_FORWARD_DL	Context – forward to dispatch level.
0x739	1849	0x9811 0739	ADSERR_DEVICE_FORWARD_RT	Context – forward to real-time.
0x740	1856	0x9811 0740	ADSERR_CLIENT_ERROR	Client error.
0x741	1857	0x9811 0741	ADSERR_CLIENT_INVALIDPARG	Service contains an invalid parameter.
0x742	1858	0x9811 0742	ADSERR_CLIENT_LISTEMPTY	Polling list is empty.
0x743	1859	0x9811 0743	ADSERR_CLIENT_VARUSED	Var connection already in use.
0x744	1860	0x9811 0744	ADSERR_CLIENT_DUPLINVOKEID	The called ID is already in use.
0x745	1861	0x9811 0745	ADSERR_CLIENT_SYNC_TIMEOUT	Timeout has occurred – the remote terminal is not responding in the specified ADS timeout. The route setting of the remote terminal may be configured incorrectly.
0x746	1862	0x9811 0746	ADSERR_CLIENT_W32ERROR	Error in Win32 subsystem.
0x747	1863	0x9811 0747	ADSERR_CLIENT_TIMEOUTINVALID	Invalid client timeout value.
0x748	1864	0x9811 0748	ADSERR_CLIENT_PORTNOTOPEN	Port not open.
0x749	1865	0x9811 0749	ADSERR_CLIENT_NOAMSADDR	No AMS address.
0x750	1872	0x9811 0750	ADSERR_CLIENT_SYNCINTERNAL	Internal error in Ads sync.
0x751	1873	0x9811 0751	ADSERR_CLIENT_ADDHASH	Hash table overflow.
0x752	1874	0x9811 0752	ADSERR_CLIENT_REMOVEHASH	Key not found in the table.
0x753	1875	0x9811 0753	ADSERR_CLIENT_NOMORESVM	No symbols in the cache.
0x754	1876	0x9811 0754	ADSERR_CLIENT_SYNCRESINVALID	Invalid response received.
0x755	1877	0x9811 0755	ADSERR_CLIENT_SYNCPORTLOCKED	Sync Port is locked.

RTime error codes

Hex	Dec	HRESULT	Name	Description
0x1000	4096	0x9811 1000	RTERR_INTERNAL	Internal error in the real-time system.
0x1001	4097	0x9811 1001	RTERR_BADTIMERPERIODS	Timer value is not valid.
0x1002	4098	0x9811 1002	RTERR_INVALIDTASKPTR	Task pointer has the invalid value 0 (zero).
0x1003	4099	0x9811 1003	RTERR_INVALIDSTACKPTR	Stack pointer has the invalid value 0 (zero).
0x1004	4100	0x9811 1004	RTERR_PRIOEXISTS	The request task priority is already assigned.
0x1005	4101	0x9811 1005	RTERR_NOMORETCB	No free TCB (Task Control Block) available. The maximum number of TCBs is 64.
0x1006	4102	0x9811 1006	RTERR_NOMORESEMAS	No free semaphores available. The maximum number of semaphores is 64.
0x1007	4103	0x9811 1007	RTERR_NOMOREQUEUES	No free space available in the queue. The maximum number of positions in the queue is 64.
0x100D	4109	0x9811 100D	RTERR_EXTIRQALREADYDEF	An external synchronization interrupt is already applied.
0x100E	4110	0x9811 100E	RTERR_EXTIRQNOTDEF	No external sync interrupt applied.
0x100F	4111	0x9811 100F	RTERR_EXTIRQINSTALLFAILED	Application of the external synchronization interrupt has failed.
0x1010	4112	0x9811 1010	RTERR_IRQNOTLESSOREQUAL	Call of a service function in the wrong context
0x1017	4119	0x9811 1017	RTERR_VMXNOTSUPPORTED	Intel VT-x extension is not supported.
0x1018	4120	0x9811 1018	RTERR_VMXDISABLED	Intel VT-x extension is not enabled in the BIOS.
0x1019	4121	0x9811 1019	RTERR_VMXCONTROLSMISSING	Missing function in Intel VT-x extension.
0x101A	4122	0x9811 101A	RTERR_VMXENABLEFAILS	Activation of Intel VT-x fails.

TCP Winsock error codes

Hex	Dec	Name	Description
0x274C	10060	WSAETIMEDOUT	A connection timeout has occurred - error while establishing the connection, because the remote terminal did not respond properly after a certain period of time, or the established connection could not be maintained because the connected host did not respond.
0x274D	10061	WSAECONNREFUSED	Connection refused - no connection could be established because the target computer has explicitly rejected it. This error usually results from an attempt to connect to a service that is inactive on the external host, that is, a service for which no server application is running.
0x2751	10065	WSAHOSTUNREACH	No route to host - a socket operation referred to an unavailable host.
More Winsock error codes: Win32 error codes			

7.3 Error diagnosis

The following chapter provides useful information for error diagnosis, if application scenarios are not functioning as expected.

Behavior	Category	Description
Connection to message broker cannot be established	Establishing the connection	<p>Check whether the message broker can be reached in principle from the system on which the Tc3_lotBase is executed (e.g. using another MQTT client).</p> <p>If the message broker cannot be reached, check your firewall settings. For an MQTT communication, the outgoing TCP port (1883 or 8883 if TLS is used) must be open by default on the MQTT client (the computer that runs the Tc3_lotBase). On the message broker side, these ports must be open for incoming messages.</p> <p>If the Tc3_lotBase is used, the port configuration can be checked via the input parameter nHostPort at the function block <code>FB_lotMqttClient</code> [► 40].</p>
Connection to <u>AWS IoT</u> [► 12] cannot be established	Establishing the connection	<p>Check whether the AWS IoT URL can be reached in principle from the system on which the Tc3_lotBase is executed (e.g. using another MQTT client). This also provides an opportunity for verifying the certificates for the connection. If the other MQTT client cannot establish a connection, check on the AWS IoT management website whether the certificates are valid and active.</p> <p>If AWS IoT cannot be reached, check your firewall settings. For an MQTT communication with AWS IoT, the outgoing TCP port (8883) must be open by default on the MQTT client (the computer that runs the Tc3_lotBase).</p> <p>If the Tc3_lotBase is used, the port configuration can be checked via the input parameter nHostPort at the function block <code>FB_lotMqttClient</code> [► 40].</p>
Connection to <u>Azure IoT Hub</u> [► 15] cannot be established	Establishing the connection	<p>Check whether the AWS IoT URL can be reached in principle from the system on which the Tc3_lotBase is executed (e.g. using another MQTT client). This also provides an opportunity for verifying the CA certificate for the connection. If the other MQTT client cannot establish a connection, check whether the CA certificate is valid.</p> <p>If the Azure IoT Hub cannot be reached, check your firewall settings. For an MQTT communication with the Azure IoT Hub, the outgoing TCP port (8883) must be open by default on the MQTT client (the computer that runs the Tc3_lotBase).</p> <p>If the Tc3_lotBase is used, the port configuration can be checked via the input parameter nHostPort at the function block <code>FB_lotMqttClient</code> [► 40].</p>
The connection cannot be established after a CRL update. The variable <code>eConnectionState</code> shows the following value: <code>MQTT_ERR_TLS_VERIFY_FAIL</code>	Establishing the connection	<p>The certificate of the message broker was revoked. In this case, please contact the operator of the message broker for further information.</p>

Behavior	Category	Description
After using an invalid topic in the meantime (e.g. a wildcard at the wrong position "testtopic#") the connection of the MQTT client to the broker toggles	Establishing the connection	After using an invalid topic TwinCAT must be restarted. Please make sure that no invalid topics are used.
Messages do not arrive at the Azure IoT Hub [► 15]	Publish	<p>Check whether the messages are published to the correct topic. The topic structure is fixed for the Azure IoT Hub, based on a naming scheme that cannot be changed.</p> <p>Check whether your MQTT settings are valid and whether the Azure IoT Hub supports them.</p> <p>For further information consult the Beckhoff notes on establishing a connection to the Azure IoT Hub [► 15] or the MSDN documentation.</p>
Messages do not arrive at IBM Watson IoT [► 13]	Publish	<p>Check whether the messages are published to the correct topic. The topic structure is fixed for IBM Watson IoT, based on a naming scheme that cannot be changed.</p> <p>Check whether your MQTT settings are valid and whether IBM Watson IoT supports them.</p> <p>For further information consult the Beckhoff notes on establishing a connection to IBM Watson IoT [► 13] or the IBM Watson documentation.</p>
Messages content is not detected as a valid JSON message by the subscriber	Publish	<p>Check whether the JSON document you are trying to send is valid. The Tc3_JsonXml library from Beckhoff provides support for creating valid JSON documents.</p> <p>Check whether the correct length information (nPayloadSize) is transferred when the publish method is called and that not too many data (null) are located after the actual payload.</p>
No properties for the corresponding event are detected at IBM Watson IoT	Publish	Check whether the correct length information (nPayloadSize) is transferred when the publish method is called and that not too many data (null) are located after the actual payload, in which case IBM Watson may not be able to recognize the message as a valid JSON message.
During the conversion of incoming MQTT messages, Node-RED reports to the JSON function "Ignored non-object payload"	Publish	Check whether the correct length information (nPayloadSize) is transferred when the publish method is called and that not too many data (null) are located after the actual payload, in which case Node-RED may not be able to recognize the message as a valid JSON message.

7.4 Support and Service

Beckhoff and their partners around the world offer comprehensive support and service, making available fast and competent assistance with all questions related to Beckhoff products and system solutions.

Beckhoff's branch offices and representatives

Please contact your Beckhoff branch office or representative for [local support and service](#) on Beckhoff products!

The addresses of Beckhoff's branch offices and representatives round the world can be found on her internet pages: <https://www.beckhoff.com>

You will also find further documentation for Beckhoff components there.

Beckhoff Support

Support offers you comprehensive technical assistance, helping you not only with the application of individual Beckhoff products, but also with other, wide-ranging services:

- support
- design, programming and commissioning of complex automation systems
- and extensive training program for Beckhoff system components

Hotline: +49 5246 963 157
Fax: +49 5246 963 9157
e-mail: support@beckhoff.com

Beckhoff Service

The Beckhoff Service Center supports you in all matters of after-sales service:

- on-site service
- repair service
- spare parts service
- hotline service

Hotline: +49 5246 963 460
Fax: +49 5246 963 479
e-mail: service@beckhoff.com

Beckhoff Headquarters

Beckhoff Automation GmbH & Co. KG

Huelshorstweg 20
33415 Verl
Germany

Phone: +49 5246 963 0
Fax: +49 5246 963 198
e-mail: info@beckhoff.com
web: <https://www.beckhoff.com>

7.5 Cipher suites used

The TwinCAT IoT driver supports secure transmission using version 1.2 of the TLS standard. Up to this version, a cipher suite is a set of algorithms (key exchange, authentication, encryption, MAC) for secure transmission.

All cipher suites currently supported by the IoT driver are listed below. The information provided here refers to TwinCAT version 3.1.4024.12.

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

More Information:
www.beckhoff.com/tf6701.html

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Germany
Phone: +49 5246 9630
info@beckhoff.com
www.beckhoff.com

