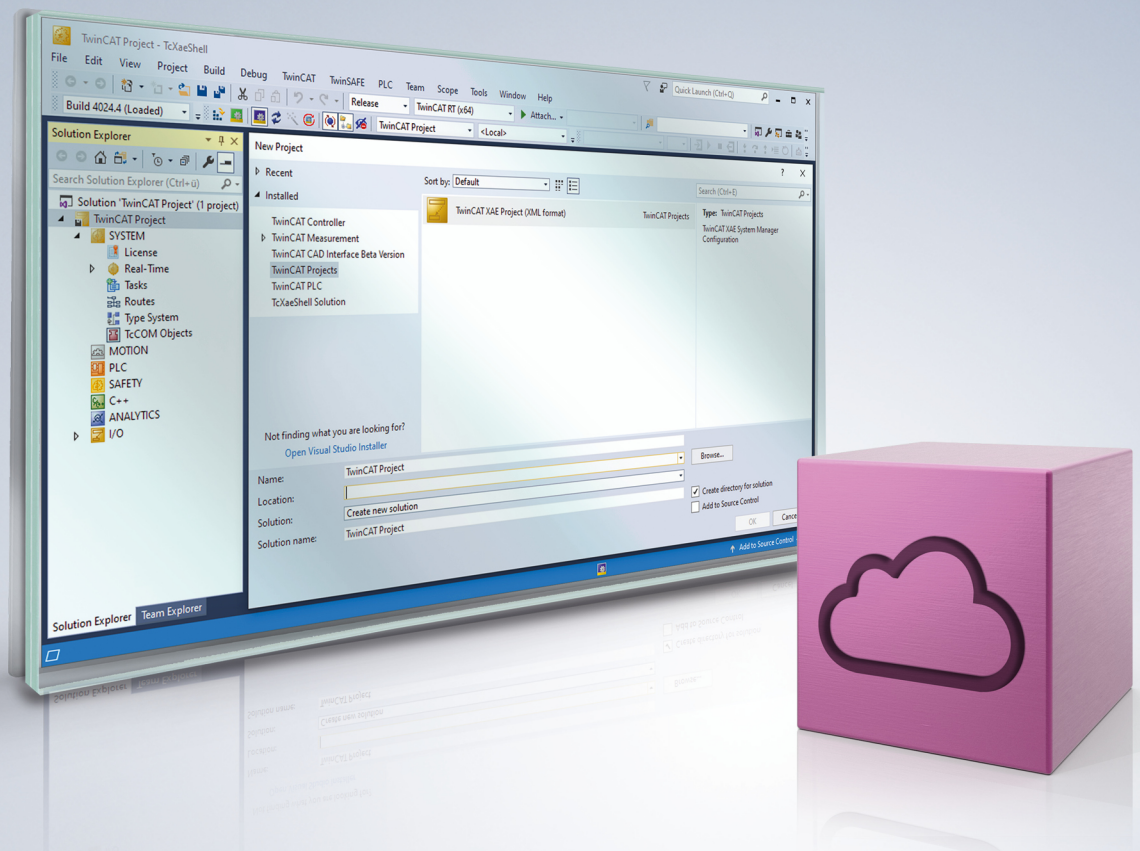
BECKHOFF New Automation Technology

Manual | EN

TF6701

TwinCAT 3 | IoT Communication (MQTT)



2024-08-28 | Version: 1.12.0

Table of contents

1 Foreword

1.1 Notes on the documentation

This description is intended exclusively for trained specialists in control and automation technology who are familiar with the applicable national standards.
For installation and commissioning of the components, it is absolutely necessary to observe the documentation and the following notes and explanations.
The qualified personnel is obliged to always use the currently valid documentation.

The responsible staff must ensure that the application or use of the products described satisfies all 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 notice.
No claims to modify 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 and licensed trademarks of Beckhoff Automation GmbH.
If third parties make use of designations or trademarks used in this publication for their own purposes, this could infringe upon the rights of the owners of the said designations.

Patents

The EtherCAT Technology is covered, including but not limited to the following patent applications and patents:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
and similar applications and registrations in several other countries.

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

Copyright

1.2 For your safety

Safety regulations

Read the following explanations for your safety.
Always observe and follow product-specific safety instructions, which you may find at the appropriate places in this document.

Exclusion of liability

All the components are supplied in particular hardware and software configurations which are 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 technology who are familiar with the applicable national standards.

Signal words

The signal words used in the documentation are classified below. In order to prevent injury and damage to persons and property, read and follow the safety and warning notices.

Personal injury warnings

⚠ DANGER

Hazard with high risk of death or serious injury.

⚠ WARNING

Hazard with medium risk of death or serious injury.

⚠ CAUTION

There is a low-risk hazard that could result in medium or minor injury.

Warning of damage to property or environment

NOTICE

The environment, equipment, or data may be damaged.

Information on handling the product

This information includes, for example:
recommendations for action, assistance or further information on the product.

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.

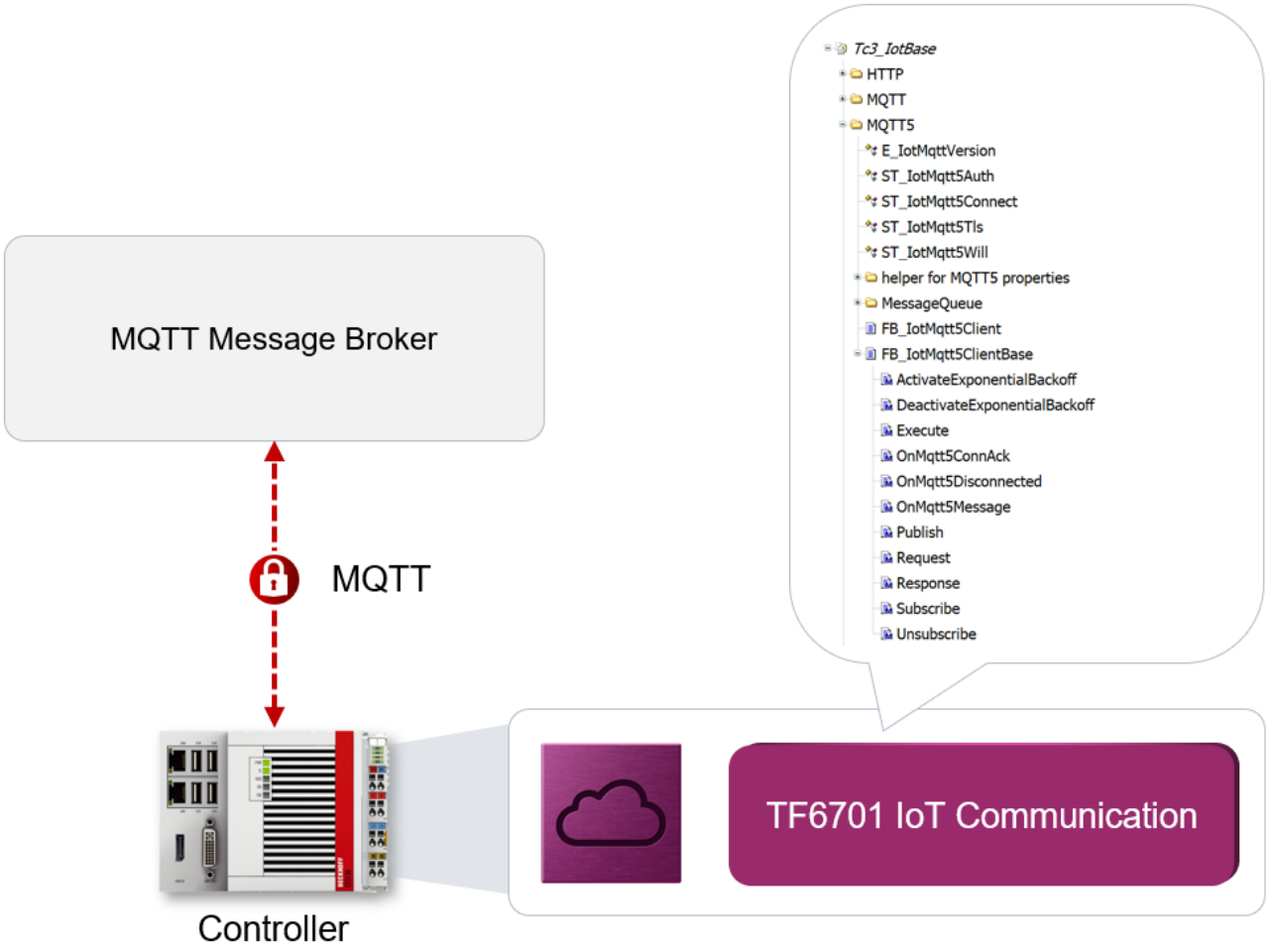
To stay informed about information security for Beckhoff products, subscribe to the RSS feed at https://www.beckhoff.com/secinfo.

1.4 Documentation output status

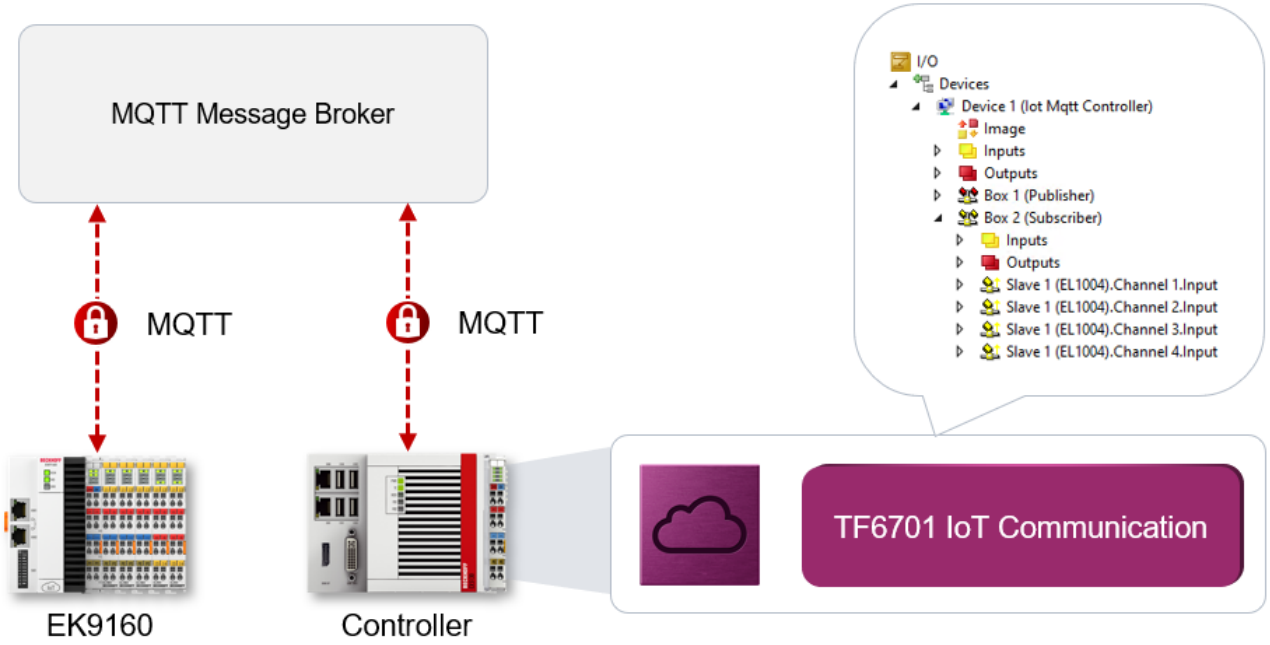
Version	Change
1.12.0	**New:** Getting started [▶ 13] Supported functions [▶ 15] Supported message brokers [▶ 17] Protocol basics [▶ 26] Security [▶ 40] GetTimeSinceLastBrokerMessage [▶ 62] MQTT5 [▶ 69]

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 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. Alternatively, an EK9160 can also be connected to a TwinCAT system via MQTT.



Product components

TF6701 IoT Communication consists of the following components, which are installed [▶ 10] with TwinCAT 3 as standard:

- **Driver**: 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, TwinCAT/BSD
Target platform	PC architecture (x86, x64 and ARM)
Minimum TwinCAT version (MQTTv3)	TwinCAT 3.1 Build 4022.0 or higher
Minimum TwinCAT version (MQTTv5)	TwinCAT 3.1 Build 4026.0 or higher
Required TwinCAT setup level	TwinCAT 3 XAE, XAR
Required TwinCAT license	TF6701 TC3 IoT Communication

Please also see our documentation article on the supported MQTT functions [▶ 15] for a fine-grained overview.

3.2 Installation

TwinCAT 3.1 Build 4022 and 4024

All the required components are supplied directly with the TwinCAT setup.

- TwinCAT XAE Setup: Contains the MQTT driver and the PLC library (Tc3_IotBase [▶ 55])
- TwinCAT XAR Setup: Contains only the MQTT driver

TwinCAT 3.1 Build 4026

The MQTT driver is already included in the TwinCAT Standard Workload. The PLC library Tc3_IotBase can be installed via the package TwinCAT.XAE.PLC.Lib.Tc3_IotBase.

If you are using TwinCAT 3.1 Build 4026 (and higher) on the Microsoft Windows operating system, you can install this function via the TwinCAT Package Manager, see Installation documentation.

Normally you install the function via the corresponding workload; however, you can also install the packages contained in the workload individually. This documentation briefly describes the installation process via the workload.

Command line program TcPkg

You can use the TcPkg **C**ommand **L**ine **I**nterface (CLI) to display the available workloads on the system:

```
tcpkg list -t workload
```

You can use the following command to install the workload of the TF6701 IoT Communication MQTT function.

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

TwinCAT Package Manager UI

You can use the **U**ser **I**nterface (UI) to display all available workloads and install them if required.
To do this, follow the corresponding instructions in the interface.

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).

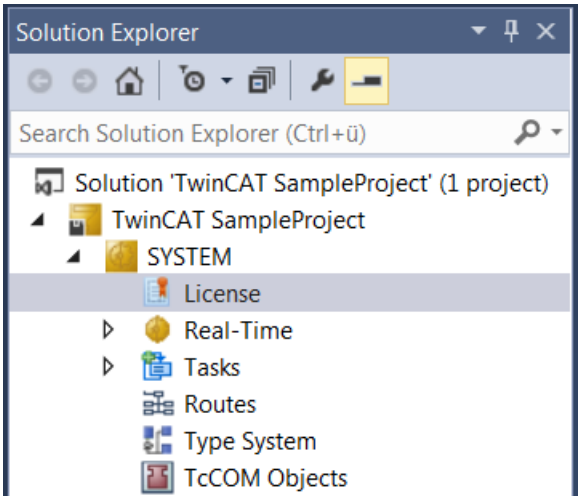
Licensing the full version of a TwinCAT 3 Function

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

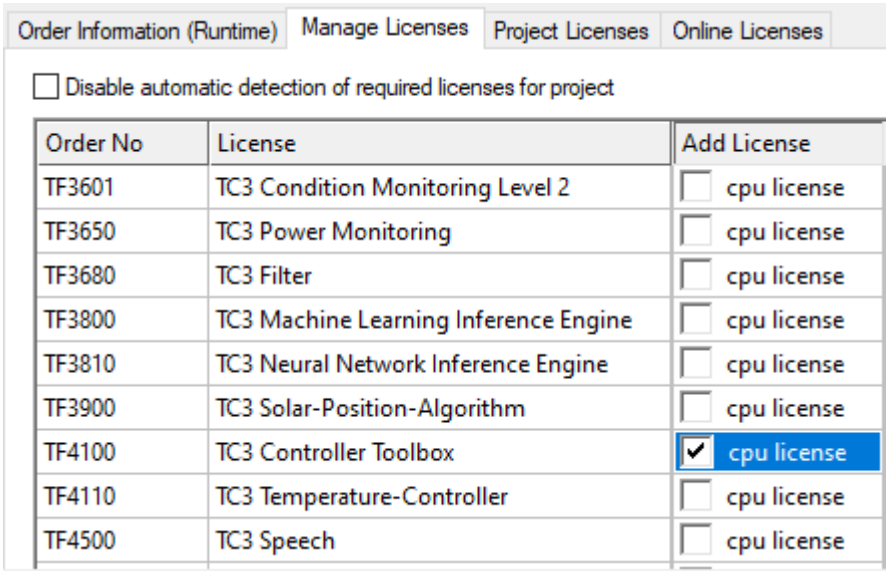
Licensing the 7-day test version of a TwinCAT 3 Function

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.



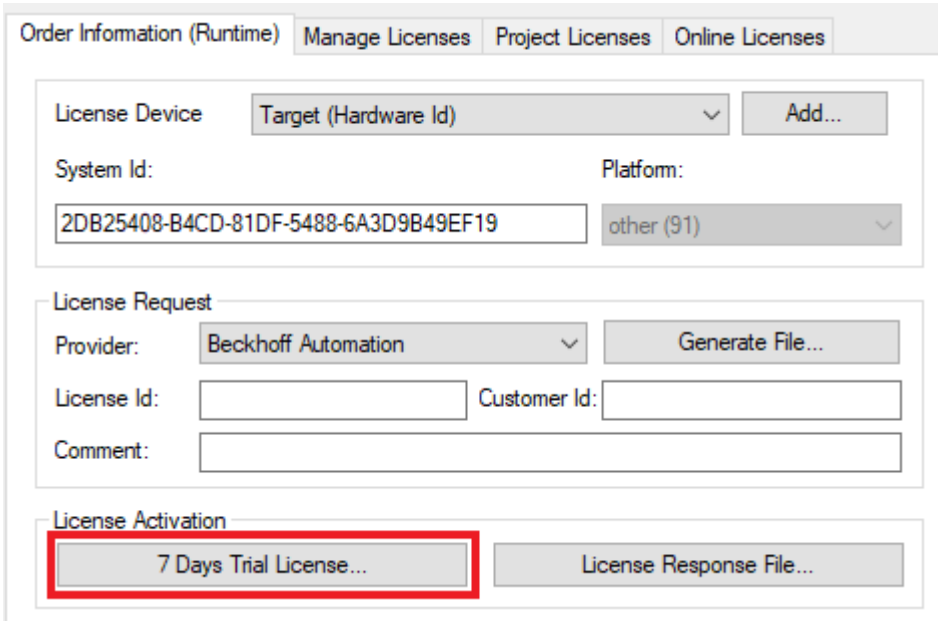
   ⇨ The TwinCAT 3 license manager opens.
5. Open the **Manage Licenses** tab. In the **Add License** column, check the check box for the license you want to add to your project (e.g. "TF4100 TC3 Controller Toolbox").



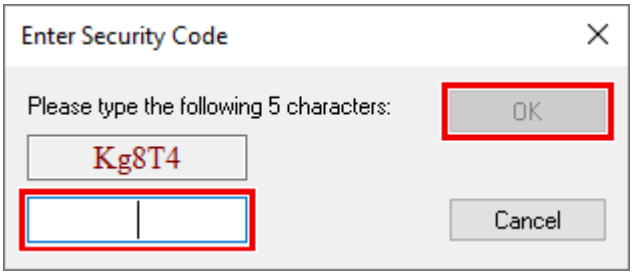
Order No	License	Add License
TF3601	TC3 Condition Monitoring Level 2	cpu license
TF3650	TC3 Power Monitoring	cpu license
TF3680	TC3 Filter	cpu license
TF3800	TC3 Machine Learning Inference Engine	cpu license
TF3810	TC3 Neural Network Inference Engine	cpu license
TF3900	TC3 Solar-Position-Algorithm	cpu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

6. 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.



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



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 Getting started

This documentation article is intended to allow you an initial, quick start in how to use this product. After the successful Installation [▶ 10] and Licensing [▶ 10], perform the following steps in order to connect to an MQTT Message Broker and send and receive messages.

Message Broker installation

This document requires the presence of a locally installed and functional MQTT Message Broker. As an example we use the Mosquitto Message Broker here, but you can use any message broker you like.

Message Broker setup

The Mosquitto Message Broker has been shipped since version 2.x with a configuration that requires security measures to be set up to ensure secure operation of the message broker. In the following, we will show you how to modify the Mosquitto Message Broker configuration so that an unsecured communication connection can be established with the broker. However, this should be done exclusively for testing purposes in a trusted operating environment. For productive operation, we recommend using a secure broker configuration.

1. Install the Mosquitto Message Broker on your system.
2. Make a backup of the mosquitto.conf file from the Mosquitto installation directory. This is typically located at C:\Program Files\mosquitto.
3. Open the mosquitto.conf file with a text editor of your choice and remove the existing content. Add the following content and save the file.

```
listener 1883
allow_anonymous true
```

4. Restart the Mosquitto Message Broker, either via the corresponding Windows service or manually via the console or mosquitto.exe.

⇨ You have now configured the Mosquitto Message Broker to listen for incoming client connections on port 1883 and it does not require any security (neither user authentication nor client certificates).

You can now start setting up the TwinCAT project.

Setup of the TwinCAT project

After you have installed the message broker, you can now start setting up the TwinCAT project to connect to the broker and exchange messages via it.

1. Create a TwinCAT project with a PLC and add Tc3_IotBase as library reference.
2. Create a program block and create an instance of FB_IotMqttClient [▶ 55] in the **declaration part** as well as two auxiliary variables to control the program flow if needed.

```
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, which is to be realized via a corresponding timer function block. A counter variable is to be used later for the message content.

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

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 in order to be evaluated sequentially. For this you declare an instance of FB_IotMqttMessageQueue [▶ 64] and an instance of FB_IotMqttMessage [▶ 66].

```
    (* 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. In this sample, the connection 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 our sample the broker is installed locally on the same system on which you want to activate the PLC project. Alternatively, you can specify the IP address or host name of the system on which the message broker was installed.

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

6. 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 (see next step) 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. In the following implementation of message evaluation, one received message is evaluated per cycle. If several messages were accumulated in the message queue, the evaluation is distributed over several cycles.

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

7. 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 this sample, `sTopicPub = sTopicSub`, so that a loopback occurs and the sent messages are received right back. In other applications the topics usually differ.

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

Next steps

Now that you have successfully connected to the message broker, we recommend that you reset the Mosquitto Message Broker to its delivery state. For this, please take the backup file of mosquitto.conf that you created in the previous steps. This file, together with the broker documentation, is a good basis for further steps, e.g. to set up a secure message broker operating environment considering client/server certificates and user authentication.

In the above sample, we have sent or received a string variable as the payload of the message. Since the data format [▶ 29] is not specified for MQTT, you are free to decide how this should be structured. However, typical IoT applications use a JSON or XML-based data format. For this purpose, Beckhoff provides the PLC library Tc3_JsonXml, which you can use to create and read JSON and XML documents.

4.2 Supported functions

The following table gives an overview of all supported functions. In general, both MQTT 3.1.1 (hereafter referred to as MQTT3) and MQTT 5.0 (hereafter referred to as MQTT5) are supported.

Function	Supported in	Min. TwinCAT version	Description
User authentication	MQTT3/MQTT5	3.1 Build 4022.0	A user name/password combination can be used to log in to the message broker.
Cipher Suites	MQTT3/MQTT5	3.1 Build 4022.0	Specification of the Cipher Suites to be used.
Clean Session	MQTT3/MQTT5	3.1 Build 4022.0	The use of CleanSession is currently supported only if CleanSession = TRUE.
Client-ID	MQTT3/MQTT5	3.1 Build 4022.0	Specification of an MQTT Client-ID.
Connection acknowledgement	MQTT5	3.1 Build 4026.0	The Connection Acknowledgement [▶ 34] feature can be used.
Data format	MQTT3/MQTT5	3.1 Build 4022.0	The data format [▶ 29] can be freely defined. Appropriate auxiliary libraries for easy use of JSON and XML are available.
Keep Alive	MQTT3/MQTT5	3.1 Build 4022.0	Set the KeepAlive value for the MQTT connection.
Last Will	MQTT3/MQTT5	3.1 Build 4022.0	Definition of a Last Will [▶ 33] message.
Message expiry interval	MQTT5	3.1 Build 4026.0	The Message expiry interval [▶ 38] can be set.
Pre-Shared Key (PSK)	MQTT3/MQTT5	3.1 Build 4022.0	A PSK can be used to secure the transport channel (using TLS).
Publish	MQTT3/MQTT5	3.1 Build 4022.0	Sending (publishing) messages to a topic.
QoS 0, 1, 2	MQTT3/MQTT5	3.1 Build 4022.0	The QoS [▶ 30] levels 0, 1 and 2 can be used for publish and subscribe operations.
Reason codes	MQTT5	3.1 Build 4026.0	Reason codes [▶ 38] can be received and evaluated.
Request/Response	MQTT5	3.1 Build 4026.0	Send/receive messages based on the Request/Response method [▶ 33].
Retain	MQTT3/MQTT5	3.1 Build 4022.0	Messages can be flagged with Retain [▶ 32] during the Publish process.
Session expiry interval	MQTT5	3.1 Build 4026.0	The Session expiry interval [▶ 38] can be set and used.
Subscribe/Unsubscribe	MQTT3/MQTT5	3.1 Build 4022.0	Create a subscription to a topic to receive messages.
Topics	MQTT3/MQTT5	3.1 Build 4022.0	Topic [▶ 27] hierarchies can be defined freely.

Function	Supported in	Min. TwinCAT version	Description
Transport Layer Security (TLS) version 1.1, 1.2 and 1.3	MQTT3/MQTT5	3.1 Build 4022.0	TLS can be used to secure the transport channel.
User Properties	MQTT5	3.1 Build 4026.0	Using User Properties [▶ 34] to define metadata on a message.
Wildcard Subscriptions	MQTT3/MQTT5	3.1 Build 4022.0	Use of wildcards (#, +) for a subscription.
Certificates	MQTT3/MQTT5	3.1 Build 4022.0	Certificates can be used to secure the transport channel (using TLS). The PEM format is used here.
Certificate revocation lists	MQTT3/MQTT5	3.1 Build 4022.0	Use of certificate revocation lists.

4.3 Supported message brokers

MQTT is a standardized transport protocol that has become widespread in recent years. Especially in the area of cloud connectivity, all major cloud providers now rely on MQTT as the transport protocol for connecting IoT devices. The following table provides an overview of common cloud services or software applications with an MQTT interface that have already been used in various customer applications. This table is not a complete listing of all supported message brokers, but only the most common systems which are often used in customer applications.

Cloud service or software	Description
AWS IoT Core	AWS IoT Core is a message broker managed by Amazon Web Services as a native cloud service. IoT devices can connect to the service via a secure transport channel and exchange data. Various mechanisms allow routing of messages to other AWS services. A corresponding Code Sample [▶ 317] demonstrates how to connect to this broker.
AWS IoT Greengrass	AWS IoT Greengrass, as part of AWS IoT Core, is a service that typically runs in an edge device environment. The integrated message broker enables MQTT communication with lower-level devices. A corresponding Code Sample [▶ 317] demonstrates how to connect to this broker.
AWS IoT Shadow	The AWS IoT Shadow stores status information of an IoT device within the AWS IoT Core service. The Code Sample for AWS IoT Core [▶ 317] can be used as a basis for communication with the AWS IoT Shadow.
Bosch IoT Suite	The Bosch IoT Suite is a product family in the Bosch IoT Cloud that enables data communication and analysis, as well as device management of IoT devices. An MQTT interface enables the reception and transmission of telemetry data. A corresponding Code Sample [▶ 322] demonstrates how to connect to this broker.
CloudMQTT Broker	Message broker as a managed cloud service. Our general Code Sample for establishing an MQTT connection [▶ 312] can be used as a basis for data communication.
IBM Watson IoT	IBM Watson IoT is a family of products in the IBM Cloud that enables data communication and analysis, as well as Device Management of IoT devices. An MQTT interface enables the reception and transmission of telemetry data. A corresponding Code Sample [▶ 323] demonstrates how to connect to this broker.
HiveMQ Message Broker	HiveMQ is a message broker that enables fast and secure data exchange via the MQTT protocol. Our general Code Sample for establishing an MQTT connection [▶ 312] can be used as a basis for data communication.
Homebridge	Homebridge allows the connection of IoT devices to Apple HomeKit, which otherwise do not have their own native Apple HomeKit support. For Homebridge there is a MQTT client plugin, through which you can connect to a message broker. Our general Code Sample for establishing an MQTT connection [▶ 312] can be used as a basis for data communication with Homebridge.

Cloud service or software	Description
MathWorks ThingSpeak	ThingSpeak is a cloud-based service that enables data aggregation, visualization and analysis of IoT data. A corresponding Code Sample [▶ 324] demonstrates how to connect to this broker.
Microsoft Azure IoT Edge	Microsoft Azure IoT Edge is a software application that typically runs in an Edge Device environment. The integrated message broker enables MQTT communication with lower-level devices. You can use our Code Sample for Azure IoT Hub [▶ 320] to establish an MQTT connection with this application.
Microsoft Azure IoT Hub	Microsoft Azure IoT Hub is a managed messaging service in the Microsoft Azure Cloud that enables secure exchange of IoT data. A corresponding Code Sample [▶ 320] demonstrates how to connect to this broker.
Microsoft Azure IoT Hub Device Twin	The Microsoft Azure IoT Hub Device Twin stores status information of an IoT device within the Microsoft Azure IoT Hub. A corresponding Code Sample [▶ 325] demonstrates how you can exchange data with the Device Twin.
Mosquitto Message Broker	Mosquitto is a message broker that enables fast and secure data exchange via the MQTT protocol. Our general Code Sample for establishing an MQTT connection [▶ 312] can be used as a basis for data communication.
Node-RED	Node-RED is a programming tool that allows graphical programming to connect devices together. Node-RED has an integrated MQTT client, which is suitable for communication with a message broker and can be connected to TwinCAT. Our general Code Sample for establishing an MQTT connection [▶ 312] can be used as a basis for data communication with Node-RED.
Shelly	Shelly devices offer an MQTT client out of the box, which can connect to a message broker and exchange data with other devices, e.g. TwinCAT. Our general code sample for establishing an MQTT connection [▶ 312] can be used as a basis for data communication with a Shelly device.

4.4 Application examples

4.4.1 Sending SMS and E-Mail notifications

Many machine applications use SMS and email notifications to send status information and alarms. This documentation describes a 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.

History

Classically, either a telephone dial-up (via a connected USB or serial modem) or an SMTP connection to a mail server directly from the machine control was used to send SMS or email messages. Although this may have worked very well, the disadvantages of this approach are quickly apparent:

- Special modem hardware and a contract with a cellular provider are required, resulting in additional costs.
- Another communication channel to a mail server is needed.

On the TwinCAT side, the TwinCAT3 Function TF6350 TC3 SMS/SMTP supports the PLC programmer in sending SMS and email messages. To send text messages, the supplement product communicates with a GSM modem connected to the controller via an RS232 serial port. For email messages, the supplement product establishes an SMTP connection to a mail server on the network. After that, the mail server is responsible for message delivery.

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.
- 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.

> **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.

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

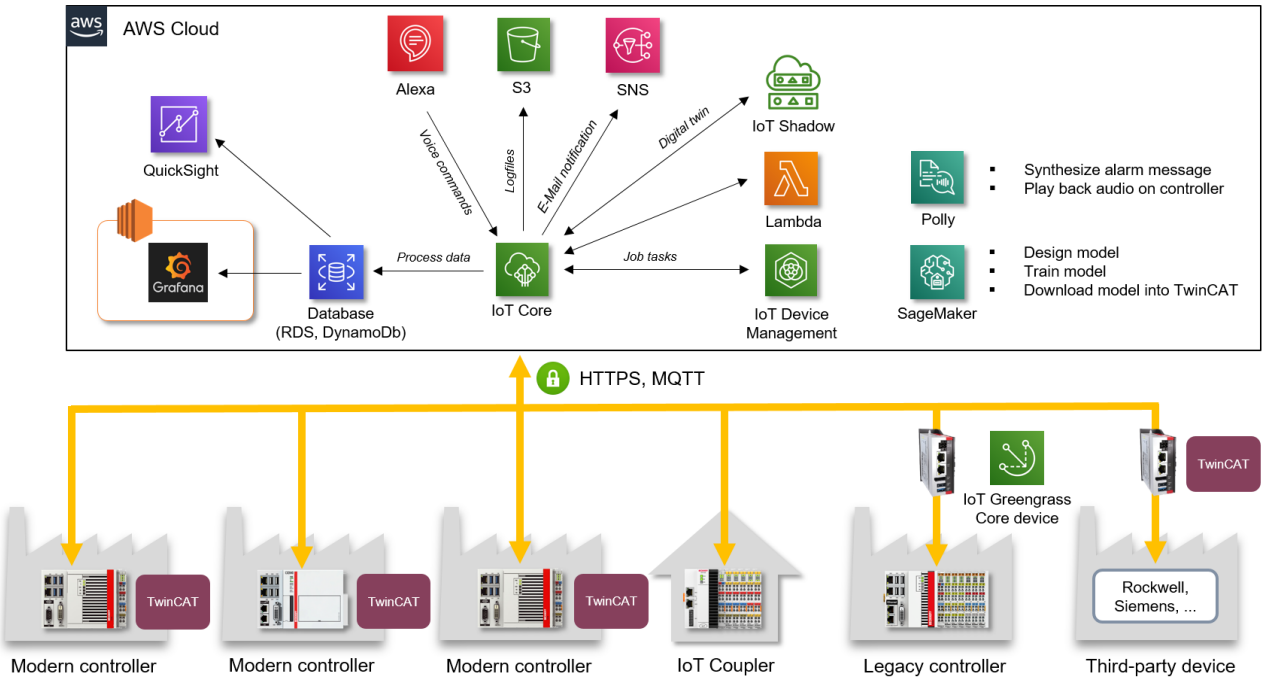
Amazon SNS: https://docs.aws.amazon.com/sns/latest/dg/sns-getting-started.html

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

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. Using so-called "rules", the user can then filter messages and easily 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.

Setup of AWS IoT Core

AWS IoT Core is a scalable and managed message broker service in the AWS Eco system. It enables you to securely connect devices and manage their data ingest. To use AWS IoT Core, you need:

- An AWS account for logging into 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 exactly the use case we are trying to solve: Configuring an AWS IoT Core rule that uses Amazon SNS to send an email or text message.

Setup of Amazon SNS

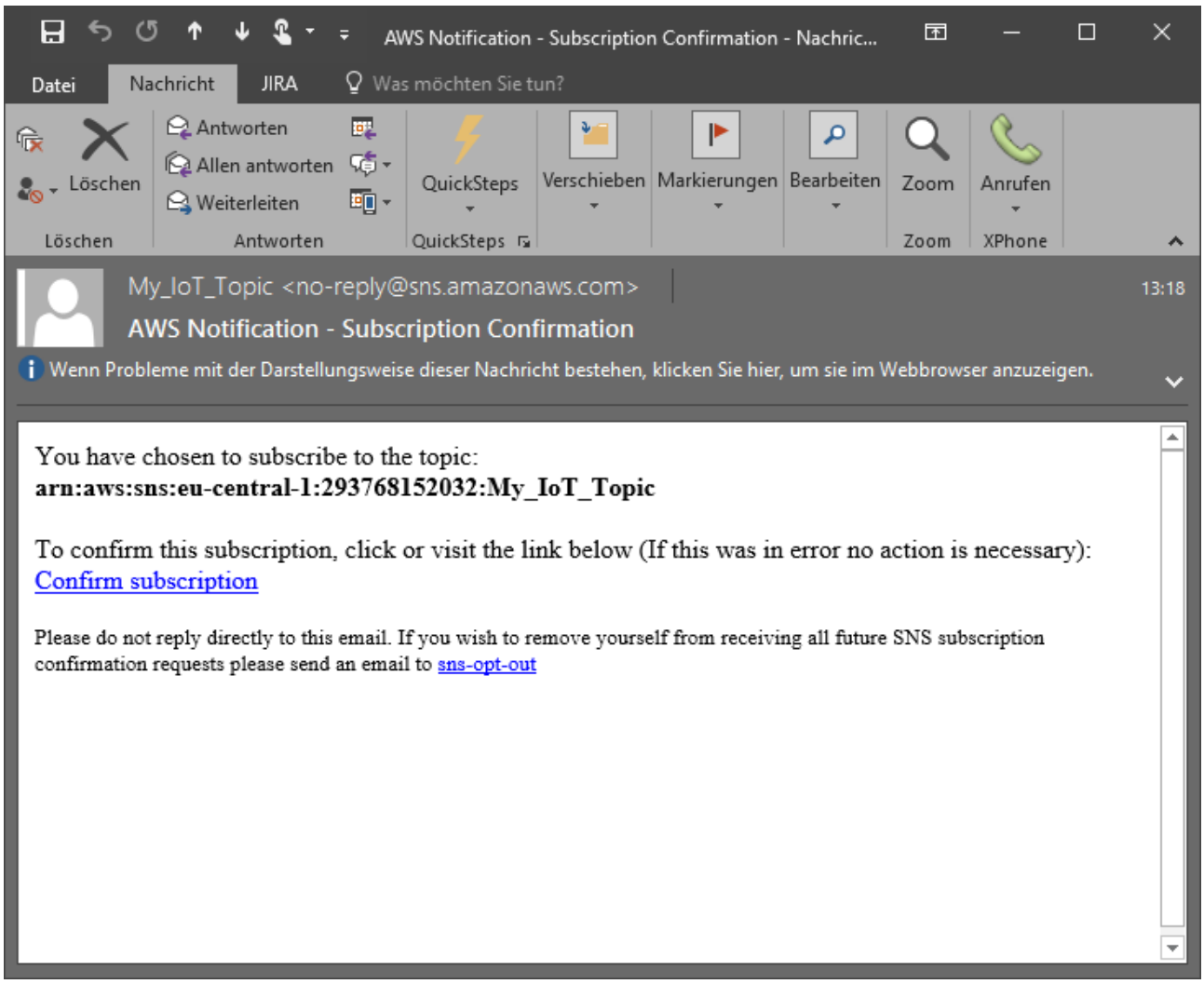
Amazon SNS allows you to send push notifications to mobile apps, text messages to mobile numbers, and plain text emails to email addresses. Step 6 ("Configuring and testing rules") of the official "Getting started with AWS IoT Core" guide describes in detail how Amazon SNS must be prepared for an AWS 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

Please 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.

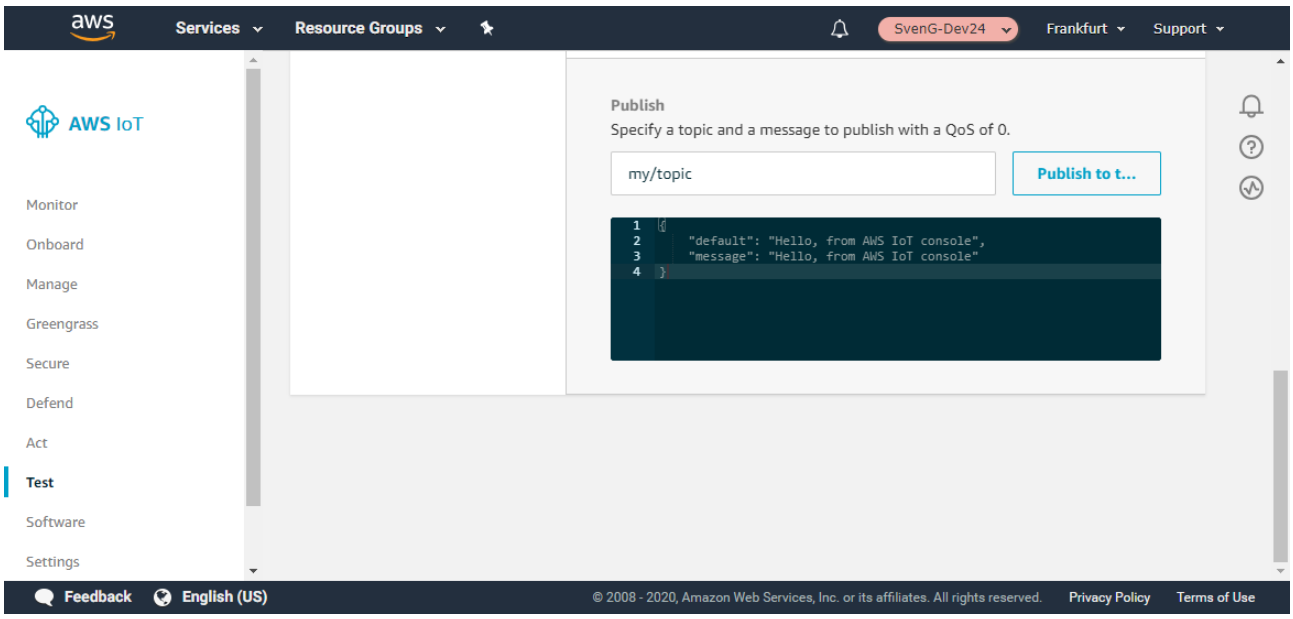


Note on service limits: note that for regular AWS accounts there may be a limit for text messages. This limit can be increased by calling AWS support. For more information, visit the Amazon SNS service limit web page.

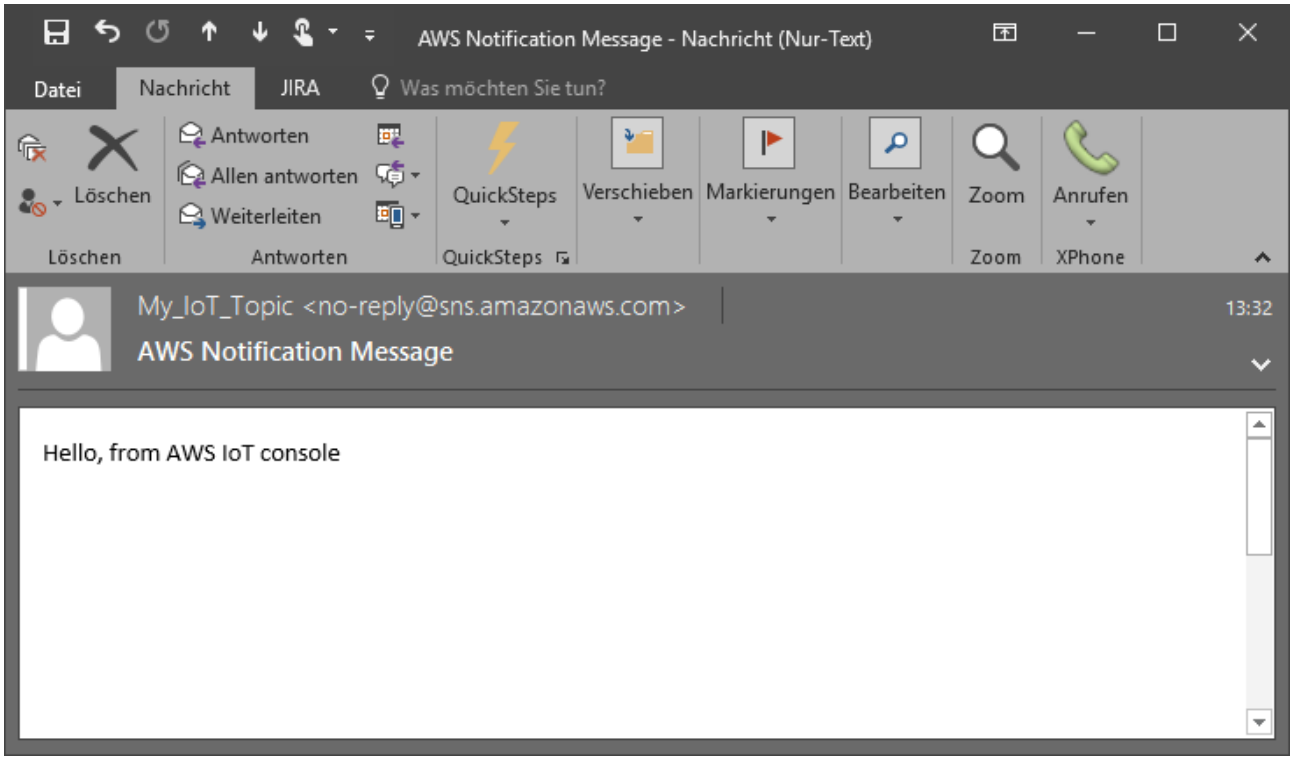
Once the Amazon SNS Topic/Subscription and the AWS IoT rule have been created, the setup can be tested using the MQTT client integrated into the AWS management console. To do this, simply send a test message to topic "my/topic" topic specified as a filter in the AWS IoT rule and use the following JSON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

Setup of 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.

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.

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

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

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.

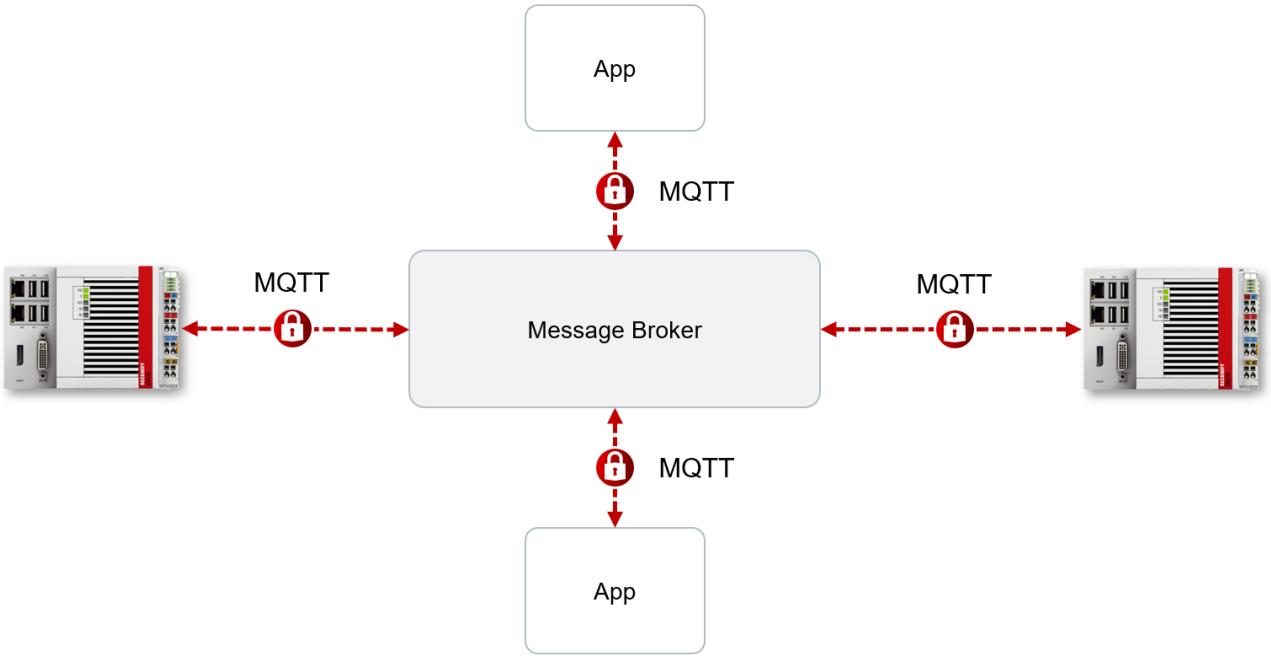
If the connection is successful, try setting bSendEmail to TRUE. After a few seconds, an email should appear in the inbox of the email address that was used for the Amazon SNS subscription.

4.5 Protocol basics

4.5.1 Overview

MQTT (Message Queueing Telemetry Transport) is a publisher/subscriber-based communication protocol, which enables message-based transfer between applications. A central component in this type of transmission is the so-called message broker, which is therefore a message-oriented middleware.

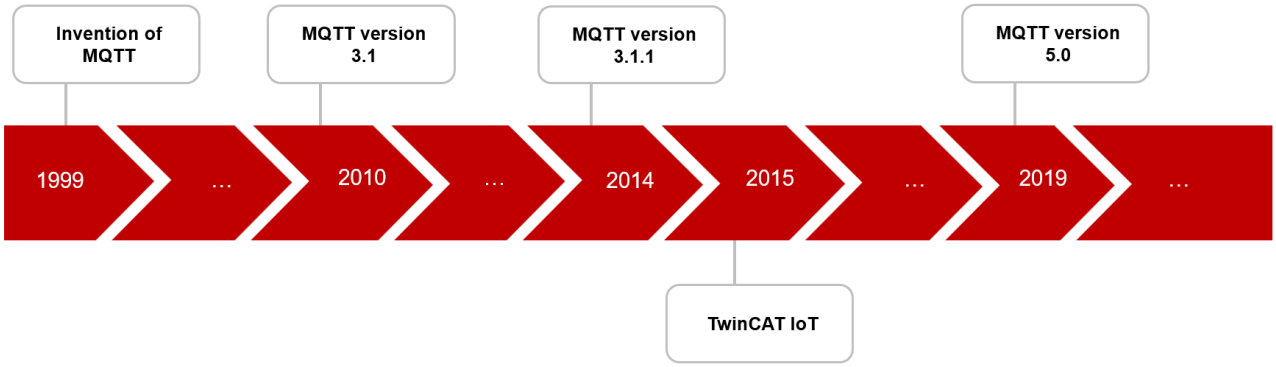
The message broker has the task of distributing messages between the individual applications, or the sender and receiver of a message. It decouples the sender and receiver, so that it is not necessary for the sender and receiver to know their respective address information. During sending and receiving all communication devices contact the message broker, which handles the distribution of the messages.



When a client connects to the message broker, security mechanisms such as TLS (Transport Layer Security) or even user name/password authentication can be used to encrypt the communication connection and realize authentication between client and message broker.

4.5.2 Protocol versions

The MQTT protocol and its specification have been available since the late 1990s. In 2019, version 5.0 of the specification was released, which adds a number of new features to the protocol.



4.5.3 ClientID

When establishing a connection with the message broker, the client transmits a so-called ClientID, which is used to uniquely identify the client on the message broker. The MQTT communication driver from TwinCAT3 automatically generates its own ClientID, which is based on the following naming scheme:

PlcProjectName-TcMqttClient%n

%n is an incremental counter for the number of the respective MQTT client instance. Each instance of the FB_IotMqttClient function block increments this counter. In most cases, using this ClientID format is sufficient. In special cases, e.g. depending on the message broker or also due to the own MQTT application, an application-specific ClientID must be assigned. This can be done via a corresponding input at the function blocks FB_IotMqttClient [▶ 55] and FB_IotMqtt5Client [▶ 79].

If a unique ClientID is to be generated automatically at the start of the PLC project, the use of a GUID is recommended, which can be generated via the function block FB_CreateGuid from the library Tc2_System. The following sample code illustrates the use of this function block.

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

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

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

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

  3: // done

255: // error state

END_CASE
```

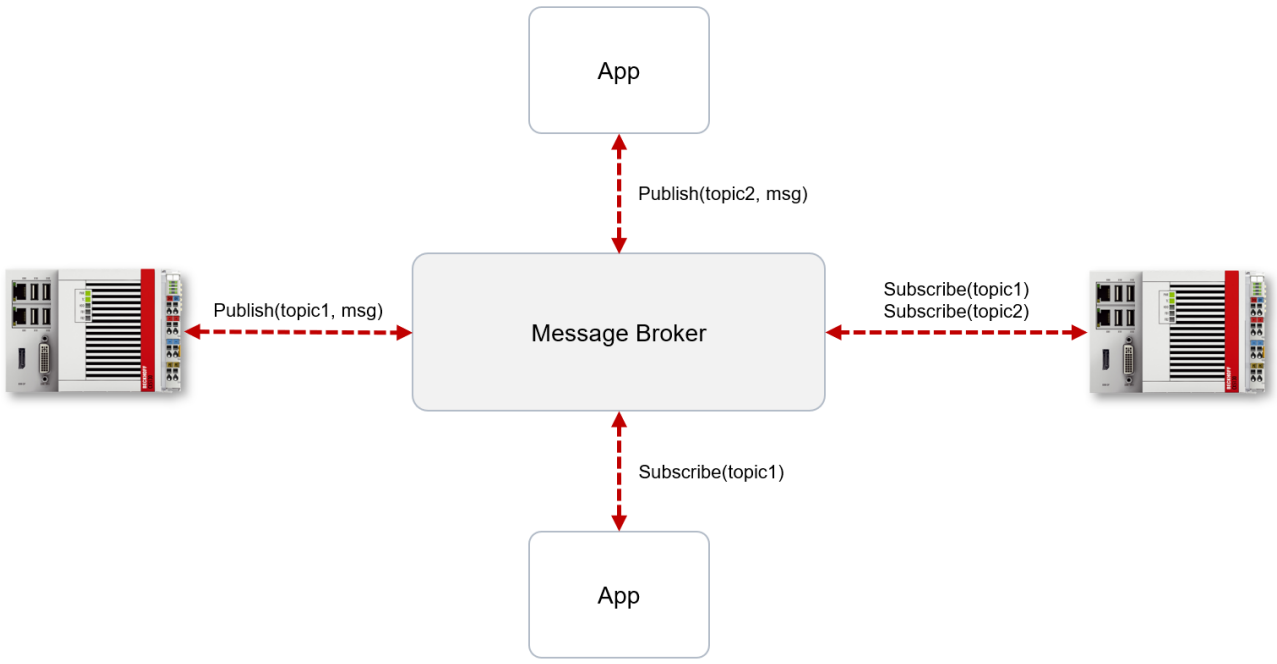
After execution of this State Machine the variable sGuid contains the generated GUID as STRING. This can then be used at the function blocks FB_IotMqttClient and FB_IotMqtt5Client as ClientID.

4.5.4 Topics

The message broker manages the addressing of messages via so-called "Topics". A topic can be thought of as a hierarchically organized mailbox, where the individual hierarchical levels are separated from each other by a "/". Sample:

```
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.



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

4.5.5 Data formats

MQTT allows the transport of binary data, whereby the actual structure of the message content is not fixed by the specification. Thus, any data can be transported, e.g. cyclic or event-based telemetry data or commands to the controller. Since the data format is not specified, the format must be known to the sender and receiver.

Over the years, a multitude of data formats and corresponding data description languages have emerged. In many IoT applications, JSON and XML have established themselves as description languages. However, the structure of a corresponding JSON or XML document is still application-specific and must be known to the applications so that they can exchange data with each other.

JSON

JSON (JavaScript Objection Notation) is a lightweight description language in an easy-to-read text form in which data is organized into so-called objects via property/value pairs. JSON has established itself in most IoT applications due to its easy readability and still low overhead (compared to other description languages such as XML). The following sample shows a JSON document containing telemetry values from three sensors including metadata and timestamps.

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

XML

XML (Extensible Markup Language) is a description language for structuring data in order to make it readable for both humans and programs. XML has a much higher overhead than JSON and is hardly used in modern IoT applications. The sample shown above could be represented in XML as follows.

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

Of course, there are many other variants of how the above JSON can be converted into an equivalent XML (and here it also becomes clear why nevertheless sender and receiver must know the data format - despite JSON or XML). Only the basic principle is to be shown here.

If you compare the two documents above, you will quickly notice the difference in size: while the JSON document has an approximate size (depending on any line breaks and spaces) of about 393 bytes, the equivalent XML document has a size of about 569 bytes and is thus about 44% larger (although the same information is transferred).

PLC library Tc3_JsonXml

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

4.5.6 QoS

QoS (Quality-of-Service) is an arrangement between the client and message broker with regard to guaranteeing of the message transfer. The QoS level must be specified by the client for both the publish and the subscribe process. There are three different QoS levels:

- 0 - message is sent at most once
- 1 - message is sent at least once
- 2 - message is sent exactly once

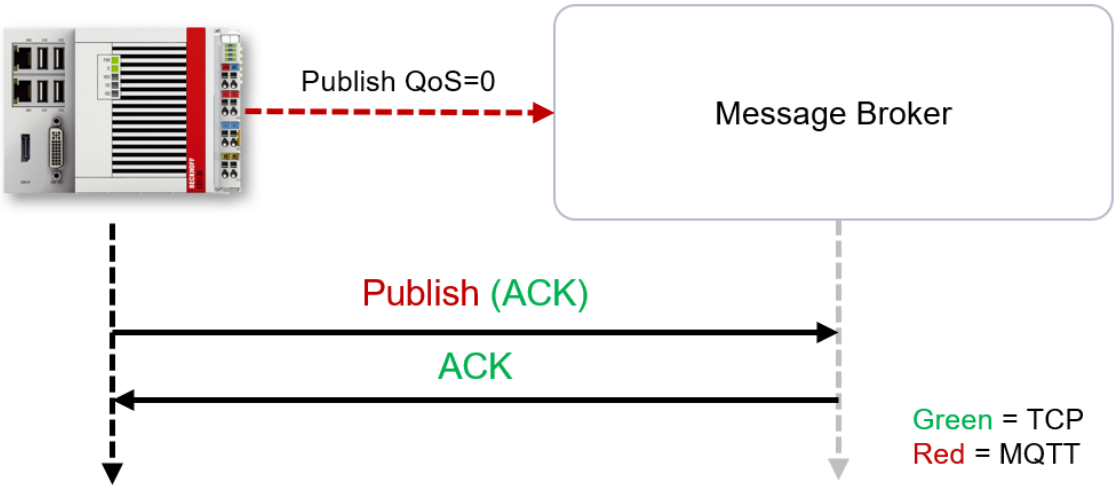
Interaction of publisher and subscriber QoS

The MQTT specification (MQTT5, chapter 3.8.4) defines a few basic rules when publisher and subscriber QoS interact, especially when publisher and subscriber specify a different QoS level.

QoS level 0

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

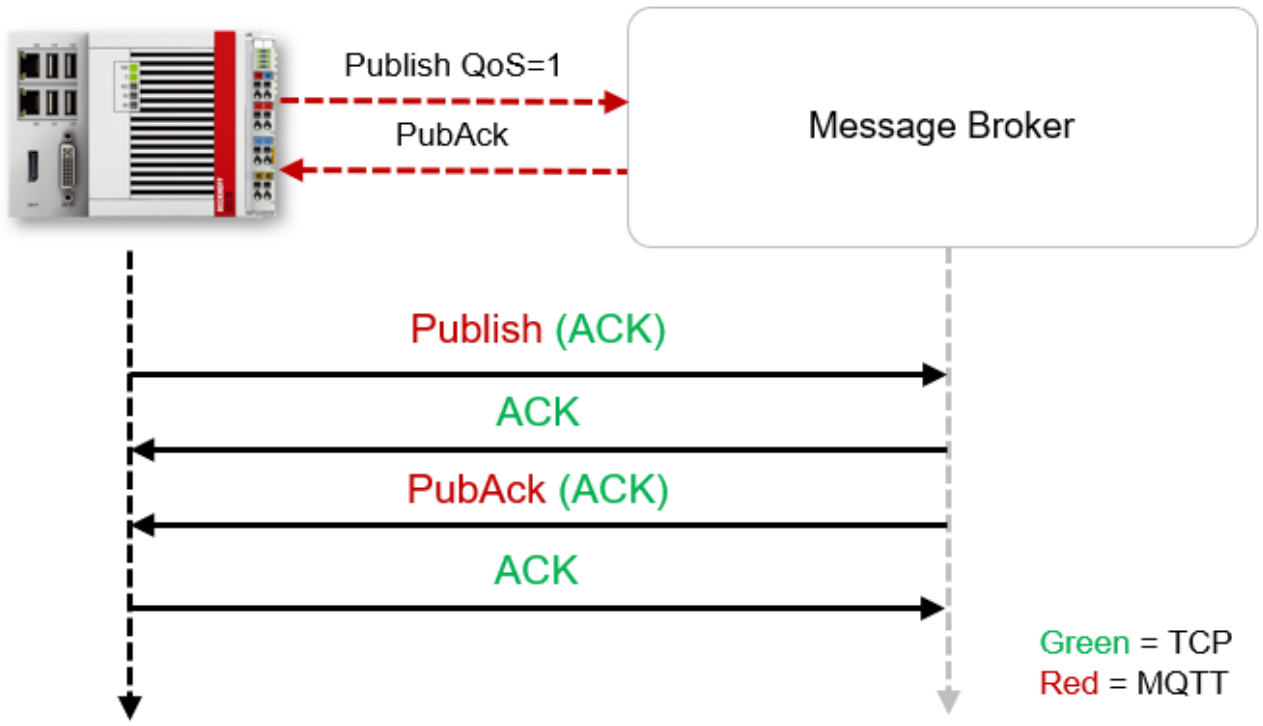
The following graphic illustrates this relationship once again, especially in comparison with the underlying TCP level.



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 acknowledgment 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.

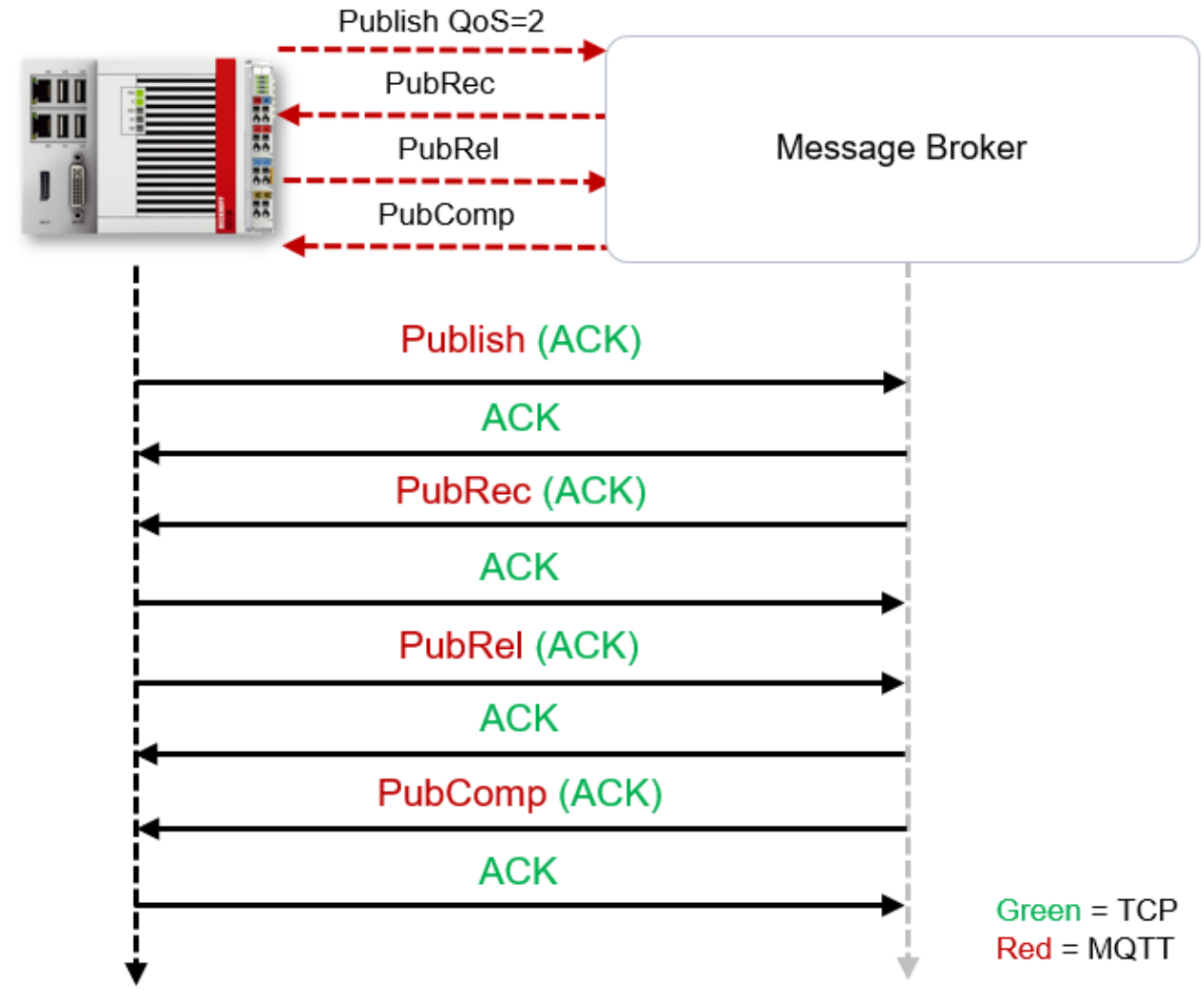
The following graphic illustrates this relationship once again, especially in comparison with the underlying TCP level.



QoS level 2

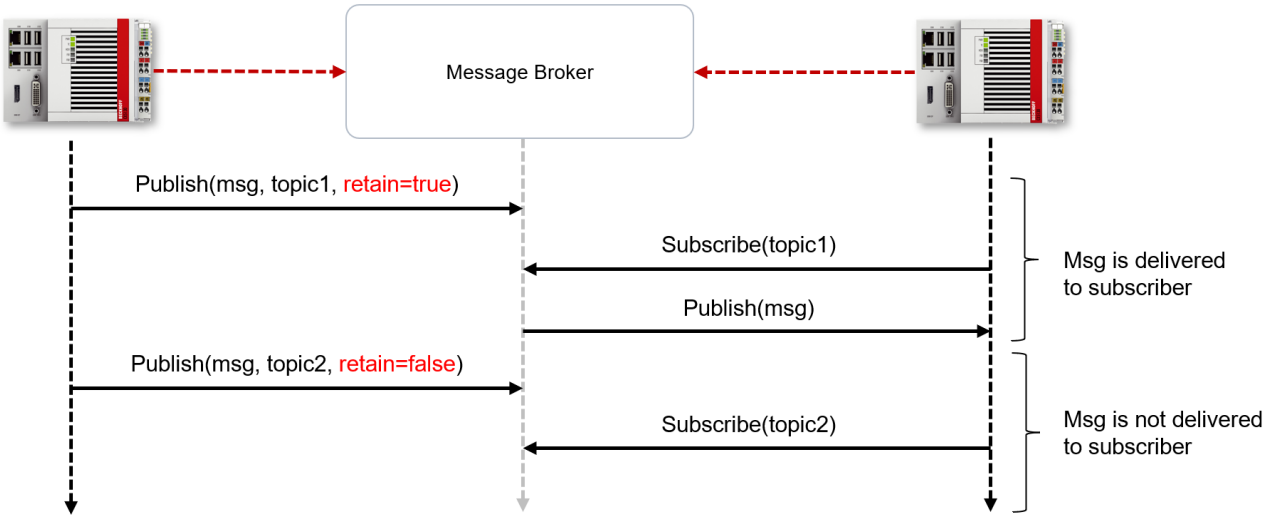
At this QoS level the system guarantees that the message arrives at the receiver no more than once. On the MQTT side this is realized through a handshake mechanism. QoS level 2 is the most secure level (from a message transfer perspective), but also the slowest. When a receiver receives a message with 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 packet is lost, the respective communication device is responsible for resending the last message after a certain time.

The following graphic illustrates this relationship once again, especially in comparison with the underlying TCP level.



4.5.7 Retain

MQTT does not define a mechanism for the publisher of a message to be guaranteed that a subscriber actually received the message. The publisher can only use the QoS [▶ 30] level to ensure that the message was delivered securely to the message broker. On the other hand, even a subscriber cannot ensure when a publisher has sent a message. In the worst case, the recipient subscribes to the topic only after the publisher has sent a message. In this case, the subscriber would not receive the message. If the retain flag is set on publish, the subscriber would receive it. The following diagram illustrates this relationship once again:



The so-called retain flag offers the possibility during the publish process to inform the message broker that it will persistently retain the message in the corresponding topic. The message broker always saves the last retain message sent. Any client that subscribes to this topic (either before or after publish) will then get this message delivered immediately once they set up the subscription. This process also works for Wildcard Subscriptions [▶ 28].

A retain message can also be removed from a topic by sending a message without payload to the corresponding topic with the retain flag set.

4.5.8 Last will

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.

4.5.9 MQTT5 extensions

4.5.9.1 Request/Response

While with MQTT3 a request/response procedure still had to be implemented manually by corresponding, applicative handshake mechanisms of the MQTT clients, MQTT5 now offers a native implementation for this. The procedure is based on the fundamental assumption that the sender of a request defines a so-called "response topic" and sends it as part of the message. The recipient of the request can then use this response topic to send its response. The following figure illustrates this process.

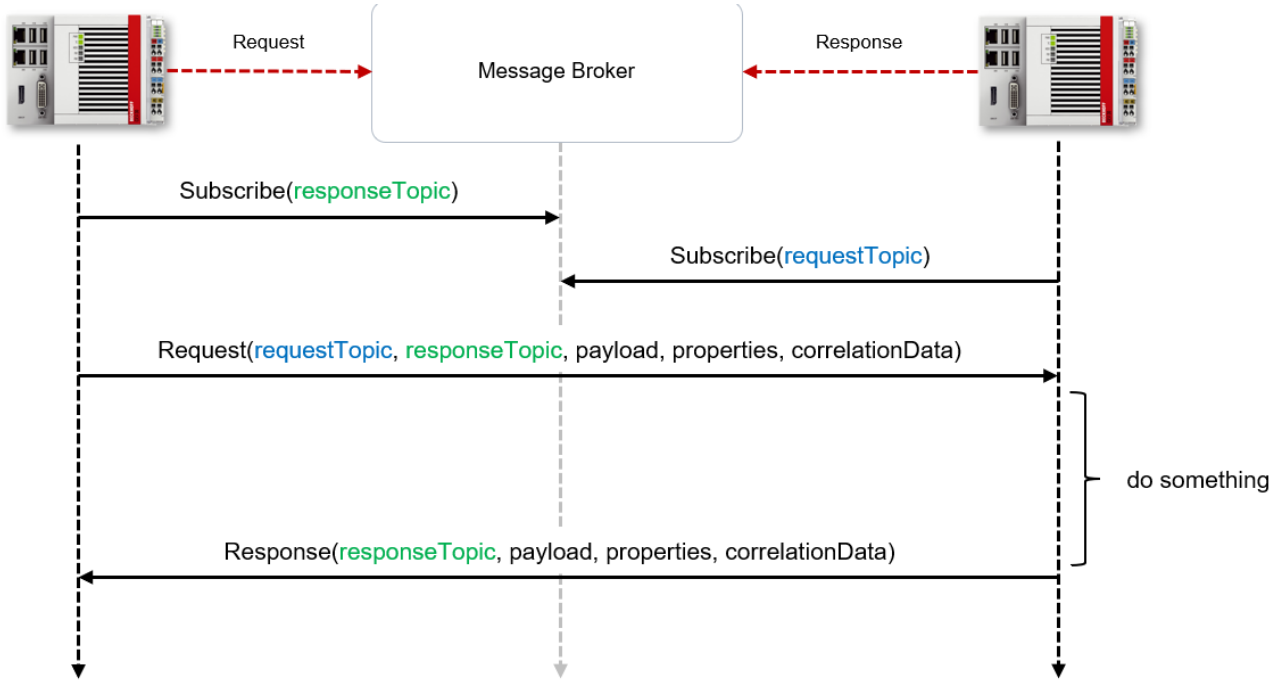


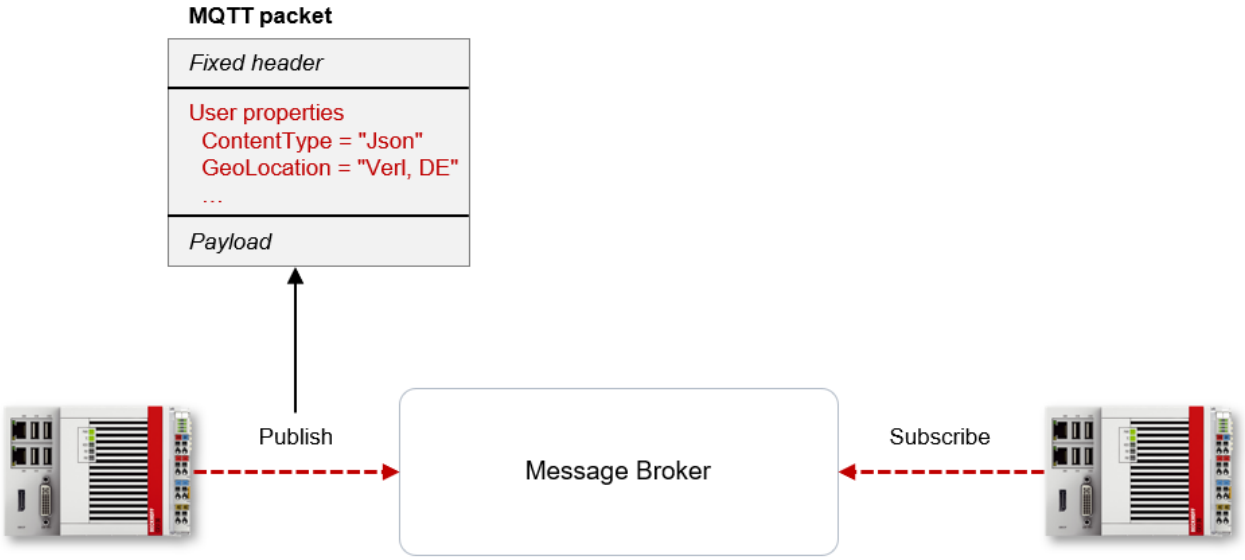
Fig. 1:

4.5.9.2 User properties

One of the most important innovations in MQTT5 are the so-called User properties. These are key/value pairs that are set for almost all MQTT packet types and can transport additional metadata - independent of the actual message content. There is no limit to the number of user properties as long as the maximum size of an MQTT message is not exceeded. This feature is similar to the key/value pairs found in an HTTP header, for example.

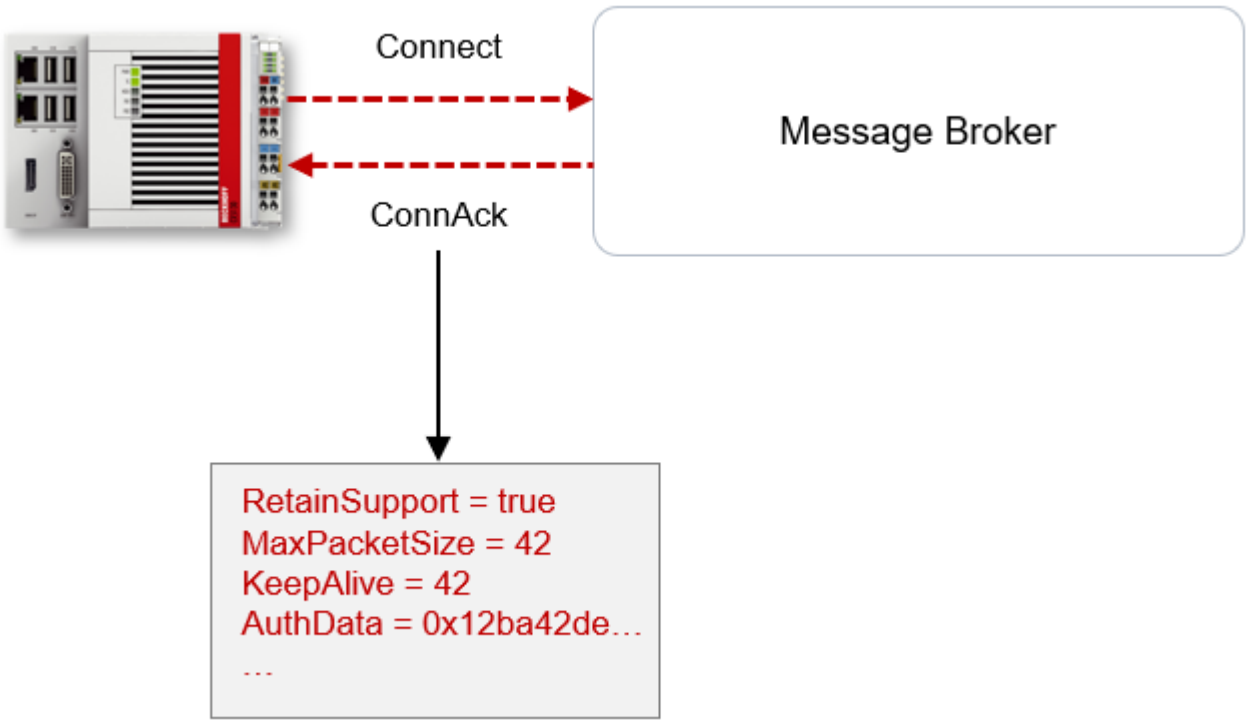
The advantage of user properties is that they are decoupled from the actual message content. For example, description information for the message content can be transmitted, which simplifies the encoding/decoding of the message content on the receiver side.

Furthermore, routing information can also be transmitted with the help of User properties, so that a recipient is told what to do with the received message content, for example.

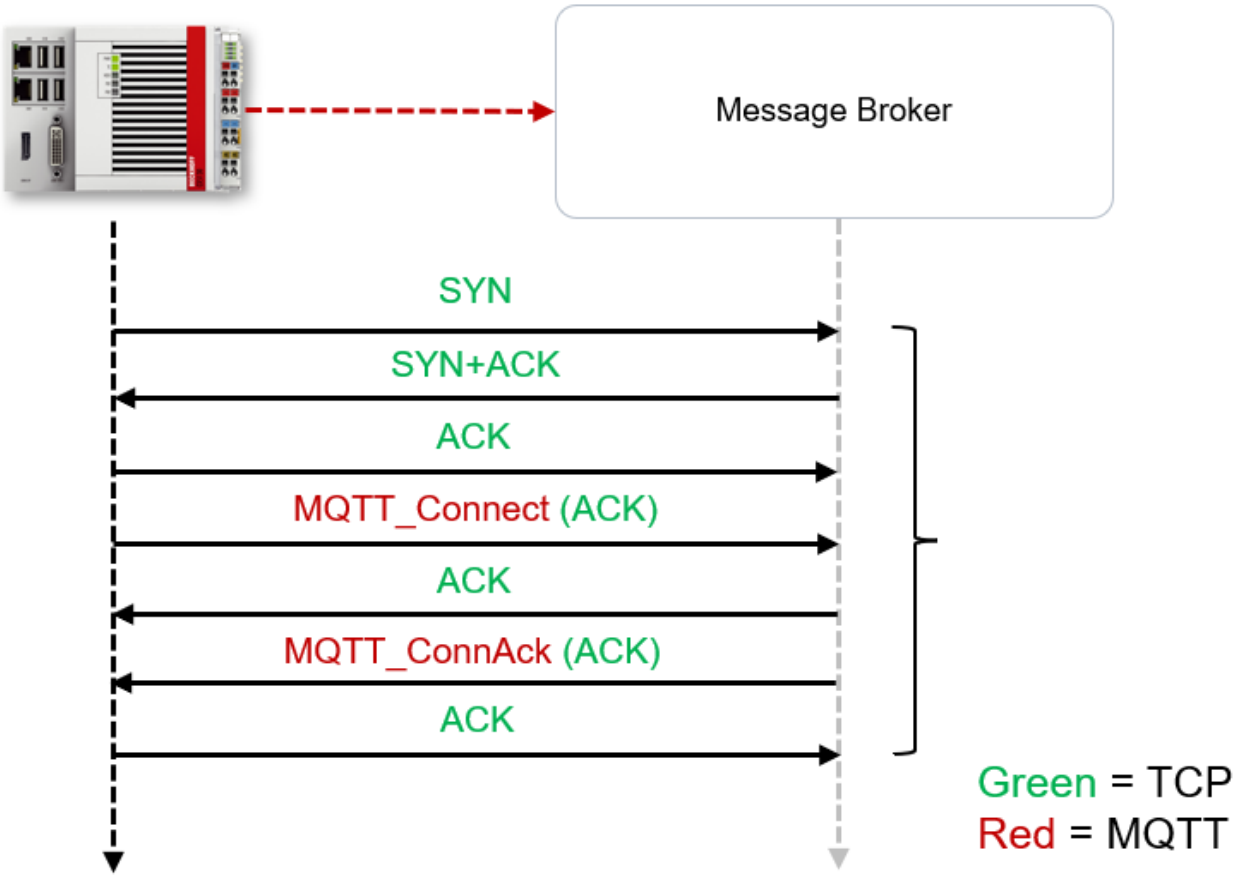


4.5.9.3 Connection acknowledgement

The newly introduced ConnAck Return Codes allow a message broker to indicate to the client which MQTT functions it supports. This information is transmitted from the message broker to the client during the ConnAck process. The following figure illustrates this process in simplified form.



The following figure shows the connection establishment between client and message broker on TCP level:

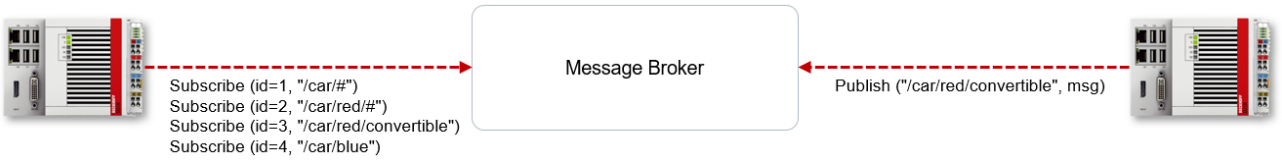


The following information can be transmitted from the message broker to the client during Connection Acknowledgement.

Property	Description
Retain available	Specifies whether the message broker supports retain messages.
Server KeepAlive	Specifies the KeepAlive time assigned by the server.
Shared subscriptions available	Specifies whether the message broker supports shared subscriptions.
Wildcard subscriptions available	Specifies whether the message broker supports wildcard subscriptions, e.g. subscriptions to a # or + topic.
Authentication data	Contains the authentication data, depending on the authentication method used.
Maximum packet size	Specifies the maximum packet size of the control packet. If this value is not specified, then no maximum size is explicitly set.
Maximum QoS	Specifies the maximum QoS level that the message broker supports. For example, some message broker implementations do not support QoS 2.
Reason code	Optional reason code that the message broker can transmit to the client as part of the ConnAck procedure. In conjunction with the Server Reference (see below), the message broker can use this to inform the client, for example, that it is temporarily or permanently unavailable and/or can be reached at a new address.
Maximum receive	The server uses this property to limit the number of QoS1 and QoS2 publishes it is willing to process for the client simultaneously.
Session expiry interval	The server can use this to inform the client that it is using a different Session Expiry Interval than originally requested by the client.
Response info	Optional return information from the message broker to the client. A use case for this can be, for example, that the broker informs the client about a "topic area" which has been assigned to the client and which it can access.
Maximum topic alias	Specifies the maximum value that can be used for a topic alias.
Assigned client ID	Returns the ClientID assigned by the message broker if the client has not specified its own ClientID.
Server reference	Optional (address) reference to another message broker, e.g. for Reason Code 0x9C or 0x9D (Server unavailable, Server has moved).
User Properties	User Properties are key/value pairs that can be attached to the PublishProperties. The meaning of the UserProperties is not part of the MQTT5 specification and therefore application specific.

4.5.9.4 Subscription identifier

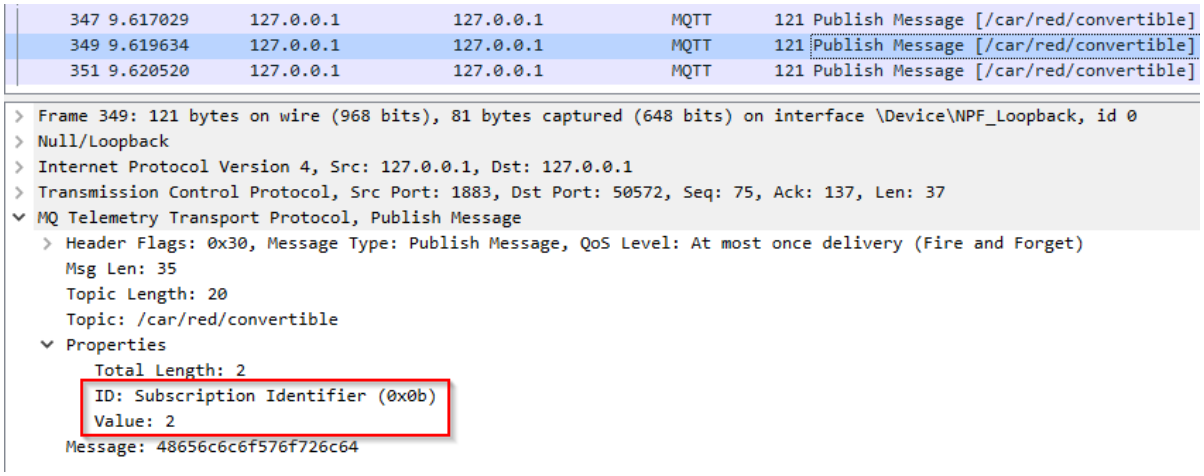
Clients can specify a so-called subscription identifier when creating a subscription. The message broker then internally stores a mapping between this subscription and the respective identifier. As soon as a message is sent from the broker to the client, the subscription identifier is also transmitted to the client. The client then has the option of assigning the message internally based on the subscription identifier. The following figure illustrates this relationship once again:



In this sample, the MQTT client creates four subscriptions on the left side, each to partially "overlapping" topics using wildcards [▶ 28]. Now the right MQTT client publishes a message on the topic /car/red/convertible.

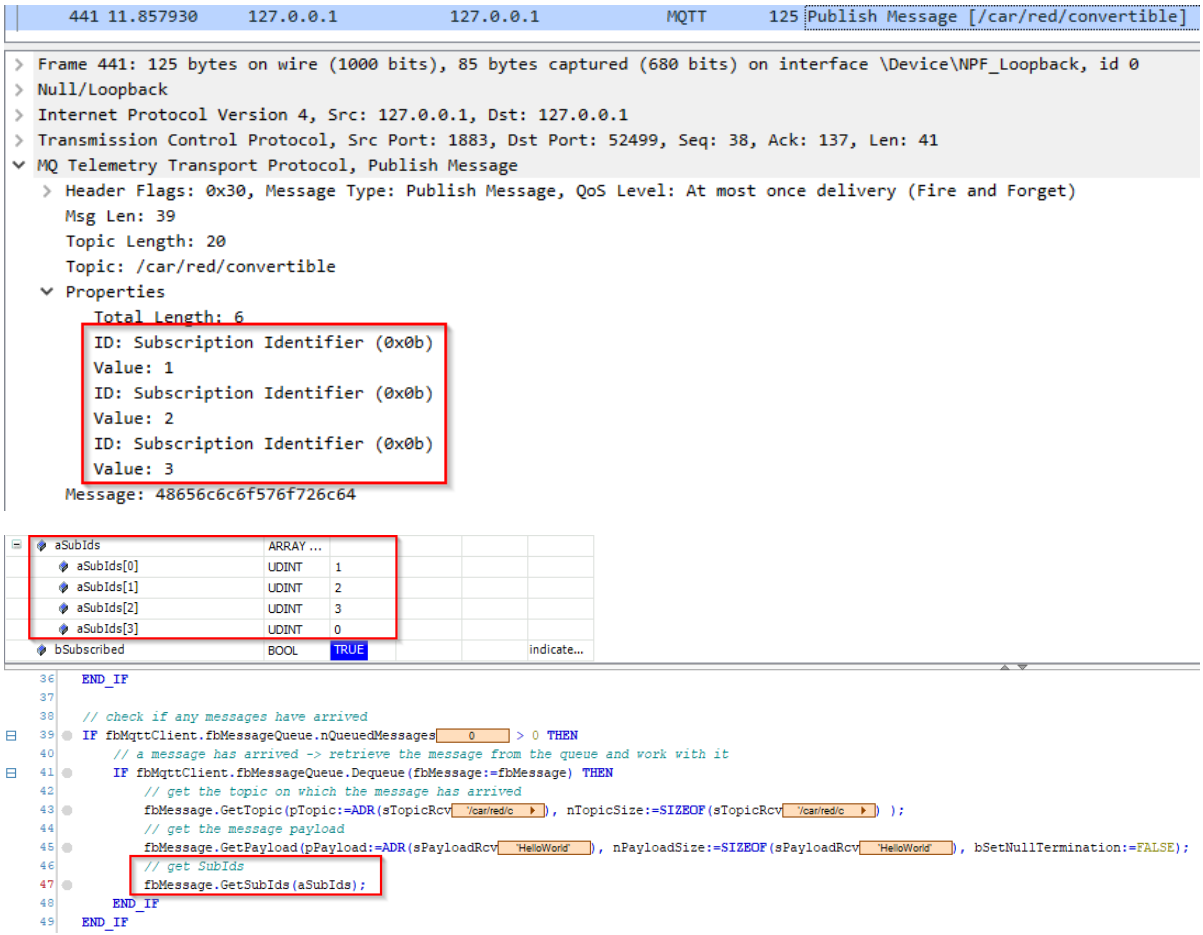
Now two options are available.

1. Unoptimized variant: The subscriber receives three MQTT messages (publishes) from the broker. Each message contains the respective subscription identifier. In the TwinCAT PLC, based on our sample IotMqttv5Sample [▶ 325], three messages would thus end up in the message queue. Example screenshot from Wireshark when receiving the three messages:



   Mosquitto Broker version 2.0.15 uses this transfer type by default.
2. The subscriber contains only one MQTT message from the broker, where this message contains a list of subscription identifiers. In the above sample, the identifiers 1, 2 and 3. In this case, only one message would end up in the message queue in the TwinCAT PLC. This message then contains an

array of the subscription identifiers. Example screenshot from Wireshark when receiving the message:



The HiveMQ CE Broker in version 2023.3 uses this transfer type by default.

Both transmission types are permitted according to the MQTT specification.

4.5.9.5 Expiry intervals

Session Expiry

With MQTT3, the Clean Session flag passed during connection establishment can be used to specify whether the message broker saves client session information after disconnection. MQTT5 replaces this setting with a Clean Start flag and the introduction of a Session Expiry interval. The Clean Start flag has the same function as Clean Session in MQTT3. However, if the client's session is not to be deleted directly, the expiry time can be set via Session Expiry (reference: ST_IotMqtt5Connect [▶ 93]).

Message Expiry

If a message with QoS 1 or 2 is published and additionally subscribed by a client with QoS 1 or 2 and the client was connected without the Clean Start flag, MQTT3 will hold a message in the message broker until the client is connected again. With MQTT5 it is possible to define an expiration time for a message. After this expiration time, the message will no longer be delivered to clients that were not connected at the time of sending.

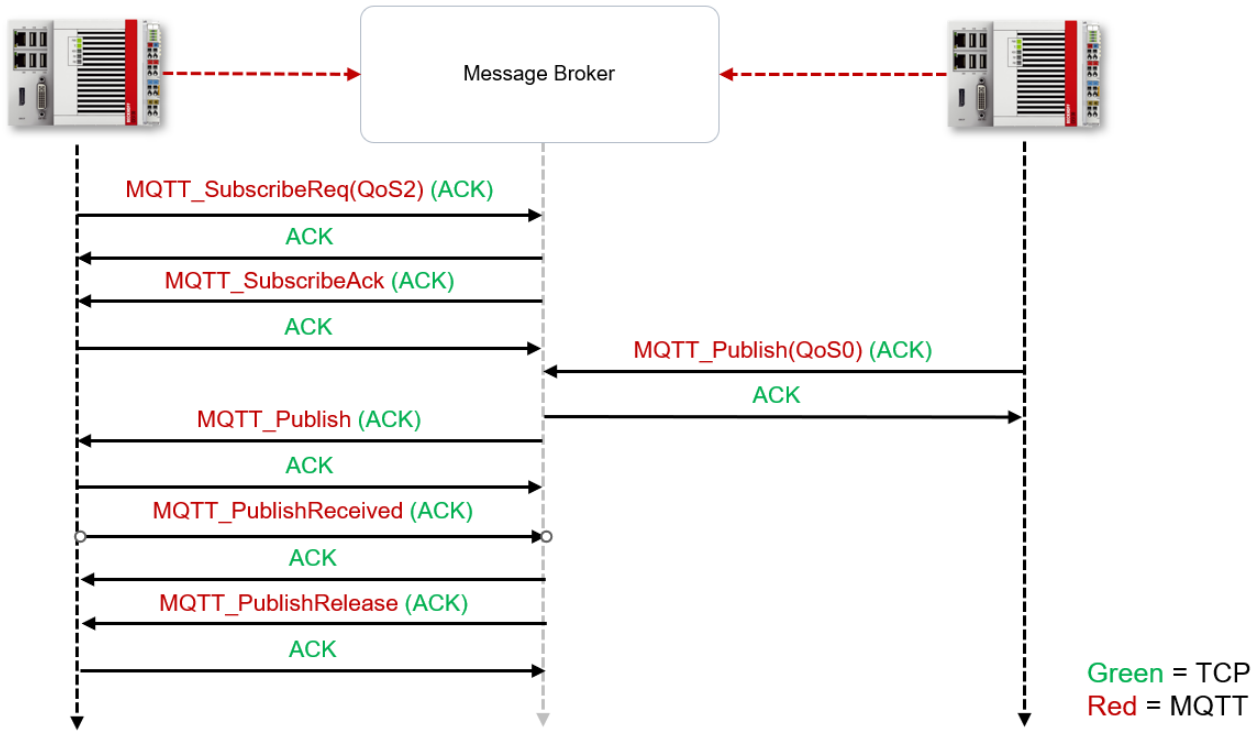
4.5.9.6 Reason codes

In MQTT3, there are only 10 Return Codes returned for the CONNACK and SUBACK message types. With MQTT5, on the other hand, it is possible to pass so-called Reason Codes in many packages. A Reason Code indicates that a predefined protocol error has occurred and is usually transmitted with acknowledge packets so that the client and message broker can react to the error condition.

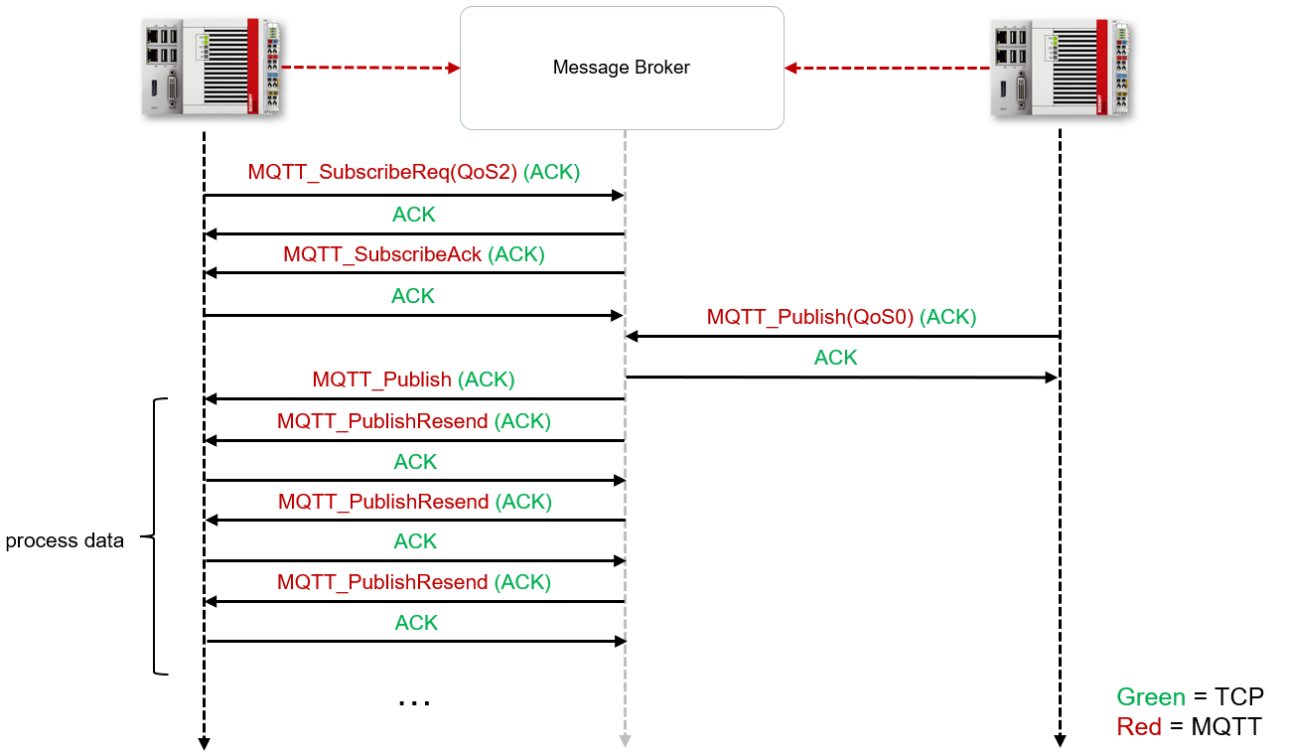
4.5.9.7 Retry-handling QoS

MQTT uses TCP as its underlying transport protocol. TCP has the property that messages arrive exactly once and in the correct order in the case of a functioning connection. All packets of an MQTT connection therefore arrive at the remote terminal. If the TCP connection is interrupted, options are available with QoS [▶ 30] 1 and 2 so that the messages can be delivered across multiple TCP connections.

MQTT3 already allowed repeated message delivery with an uninterrupted TCP connection. However, this could result in an overloaded MQTT client being further overloaded in case of doubt, e.g. if the client needs several seconds to process a received message and in the meantime the message broker sends the message again because it has not (yet) received the acknowledgement. The following figure illustrates the reception process of a message from the subscriber's point of view, which has subscribed to a topic at the message broker with QoS 2:



Now, if the receiver does not immediately send a MQTT_PublishReceived command after the MQTT_Publish due to an overload, the message broker at MQTT3 will prepare a resend of the publish and send it on its way. The TCP connection is still active in this case. This is illustrated in the following figure.



Due to a new extension in MQTT5, repeated MQTT message transmission (Resend) is prevented for clients and message brokers when a TCP connection is established.

4.5.9.8 Bi-directional disconnect

In MQTT3, a client could perform a proper Disconnect with the message broker, i.e., the client notified the broker of the disconnection. However, the protocol did not allow the message broker to inform the clients that it was disconnecting.

In MQTT5 there is now such a mechanism. The message broker can send an MQTT Disconnect packet to the clients and also specify a Reason Code for the disconnection.

4.6 Security

When considering protection of data communication, a distinction can be made between two levels: protection of the transport channel [▶ 40] and protection at application layer [▶ 47].

4.6.1 Transport layer

The worldwide common 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, taking TLS version 1.2 as an example.

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.

Supported functions

The TwinCAT IoT driver enables the use of the following TLS functions.

Function	Description
Self-signed client certificates	Use a self-signed client certificate to authenticate to the message broker.
CA-signed client certificates	Use of a CA-signed client certificate to authenticate to the message broker. The CA certificate can also be specified to establish a trust relationship.
Certificate revocation lists	Use of certificate revocation lists (CRL).
Pre-Shared Key (PSK)	Use of a pre-shared key (PSK) to authenticate to the message broker.

Cipher suite definition

A cipher suite is by definition a composition of algorithms (key exchange, authentication, encryption, MAC) for encryption. The client and server agree on these during the TLS connection establishment. For more information on cipher suites please refer to the relevant technical literature.

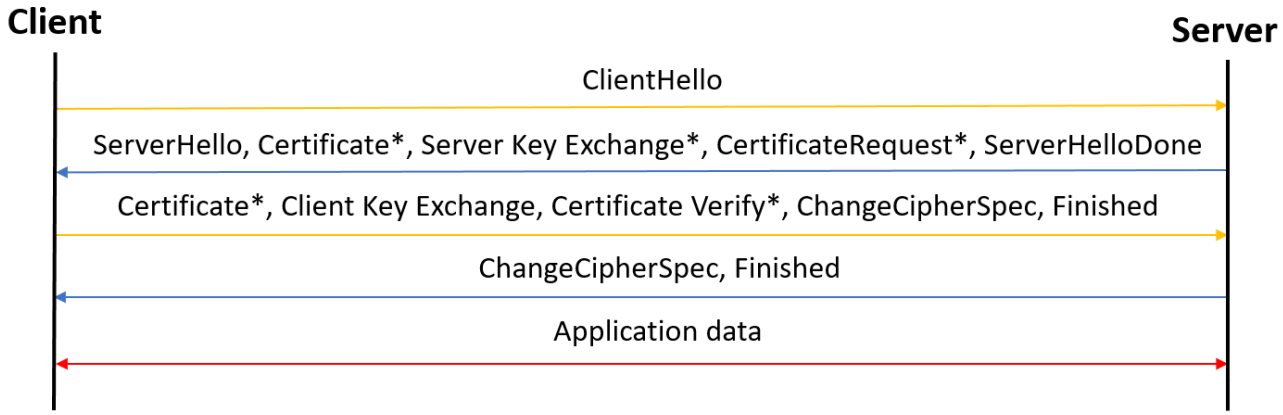
TLS communication flow

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 [▶ 43])
- The client and server identify themselves through a certificate (see Client/Server certificate [▶ 44])
- Pre-shared keys (see Pre-shared keys [▶ 44])

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.

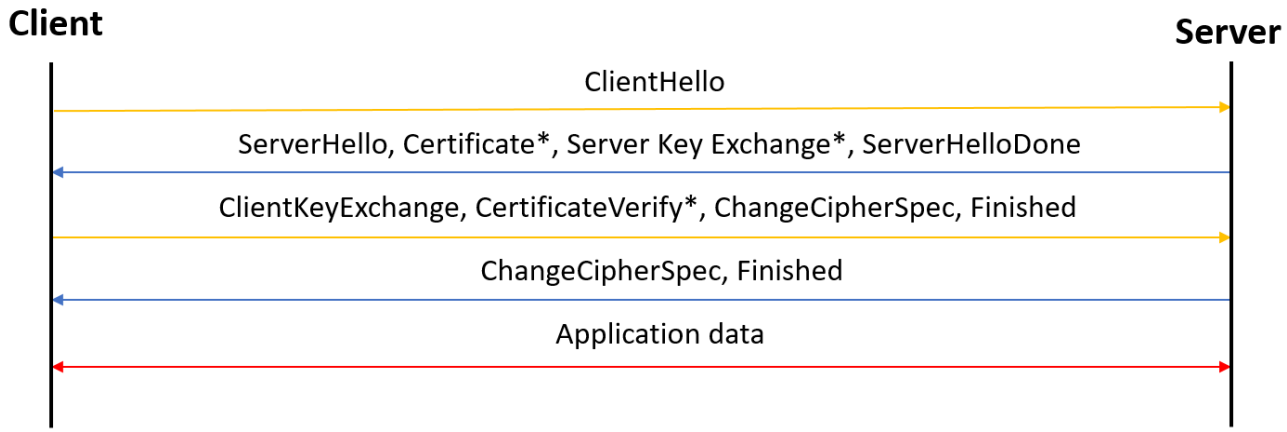
The following table explains the individual steps from the communication flow shown above.

Step	Description
ClientHello	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.
ServerHello	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.
ServerKeyExchange	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.
CertificateRequest	The server requests a certificate from the client to verify the identity of the client.
ServerHelloDone	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.
ClientKeyExchange	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.
CertificateVerify	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.
ChangeCipherSpec	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.

Step	Description
ChangeCipherSpec	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. Here too, if the hash and MAC are not successfully decrypted, the connection is aborted.
ApplicationData	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 [▶ 40] 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 (.PEM or .DER file) or the content of the .PEM file is specified 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, the MQTT client and the WebSocket client. The HTTP client is used as an example.

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

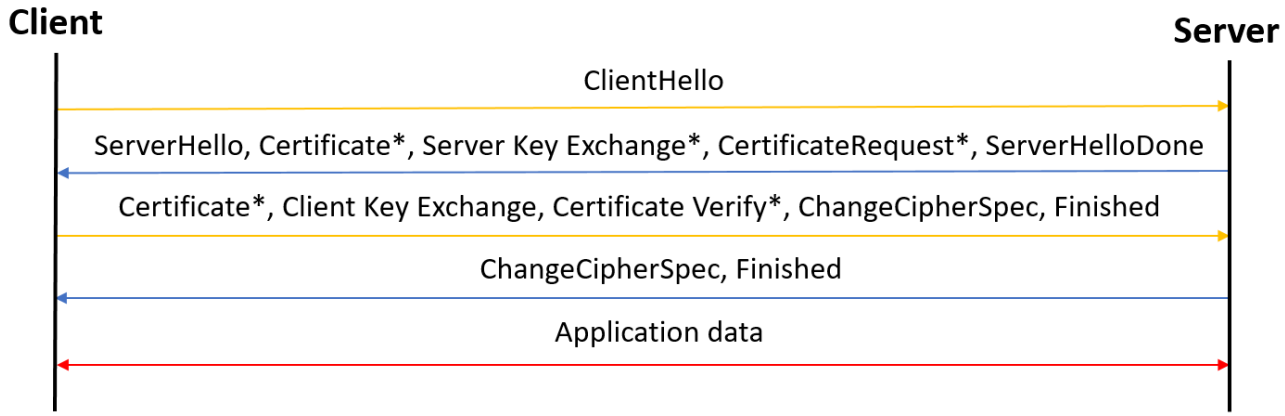
```
fbClient.stTLS.sCA:= 'C:\TwinCAT\3.1\Config\Certificates\someCA.pem';
```

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 [▶ 43] chapter) is visualized in the following diagram. The individual steps of the TLS connection establishment are described in chapter Transport layer [▶ 40].



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, the MQTT client and the WebSocket client. The HTTP client is used as an example.

```
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';
```

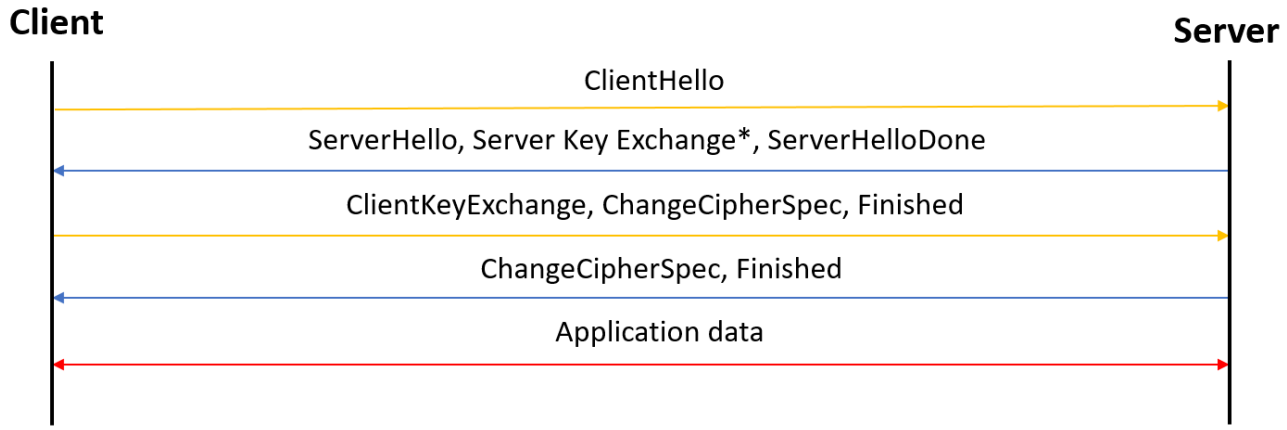
If a client certificate is set, a CA certificate must also be set to validate the server certificate. This is due to the behavior of the security framework used in the IoT driver.

If the validation of the server certificate is to be shutdown in this case, an additional flag can be set to skip the validation. However, it is not possible to omit the CA certificate.

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 where CPU power is limited. 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, the MQTT client and the WebSocket client. The HTTP client is used as an example.

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

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

4.6.1.4 Supported cipher suites

The TwinCAT IoT driver supports secure data transmission using the TLS standard. Below you will find an overview of all cipher suites supported by the IoT driver, depending on the TwinCAT version.

TwinCAT 3.1 Build 4024.x

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

TwinCAT 3.1 Build 4026.x

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

4.6.2 Application level

User identity

The MQTT specification only defines a user name/password authentication at the application layer, which can be done when establishing a connection. Other mechanisms are not specified and can be handled applicatively. Some message brokers also allow the CommonName attribute from the client certificate to be used as a user identity, which then allows access rights to be defined at topic level, for example.

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 JSON Web Token via the method FB_JwtEncode. The token can then be sent, for example, in the payload of the message or also in a User Property [▶ 34] to enable validation of the message at the receiving application.

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 [▶ 57] method must be called cyclically in the background (see FB_IotMqttClient [▶ 55]). 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 [▶ 57] 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.

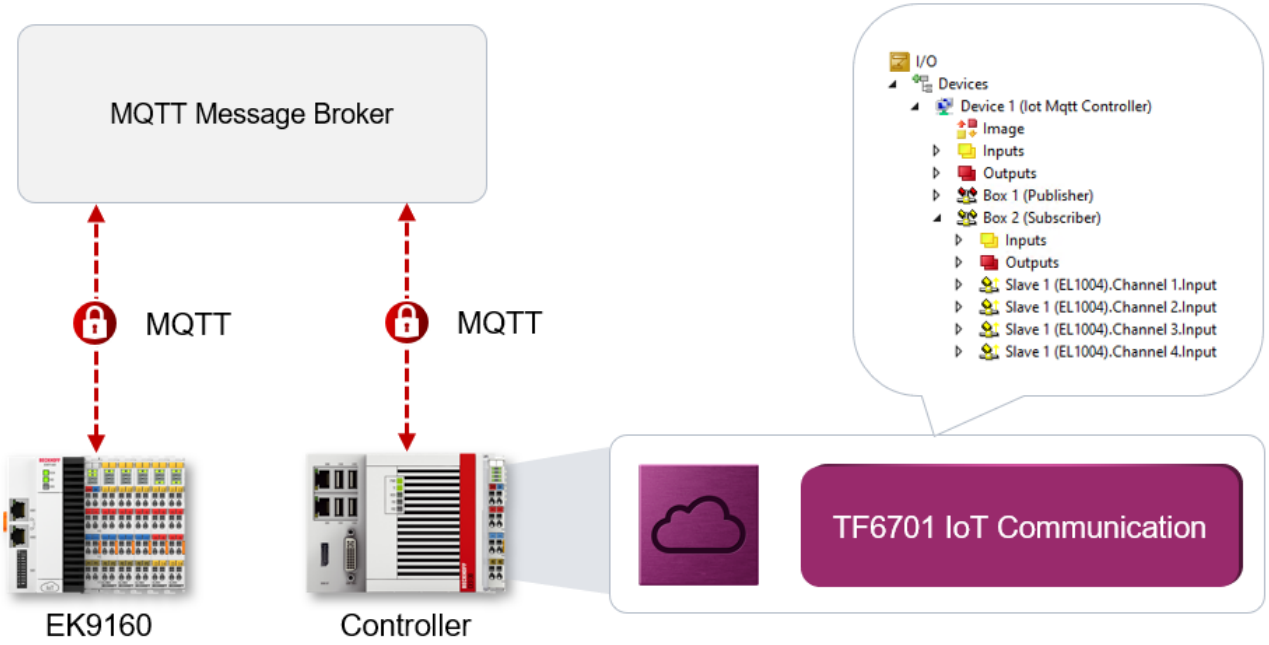
```
fbMqttClient.Execute(bConnect);

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

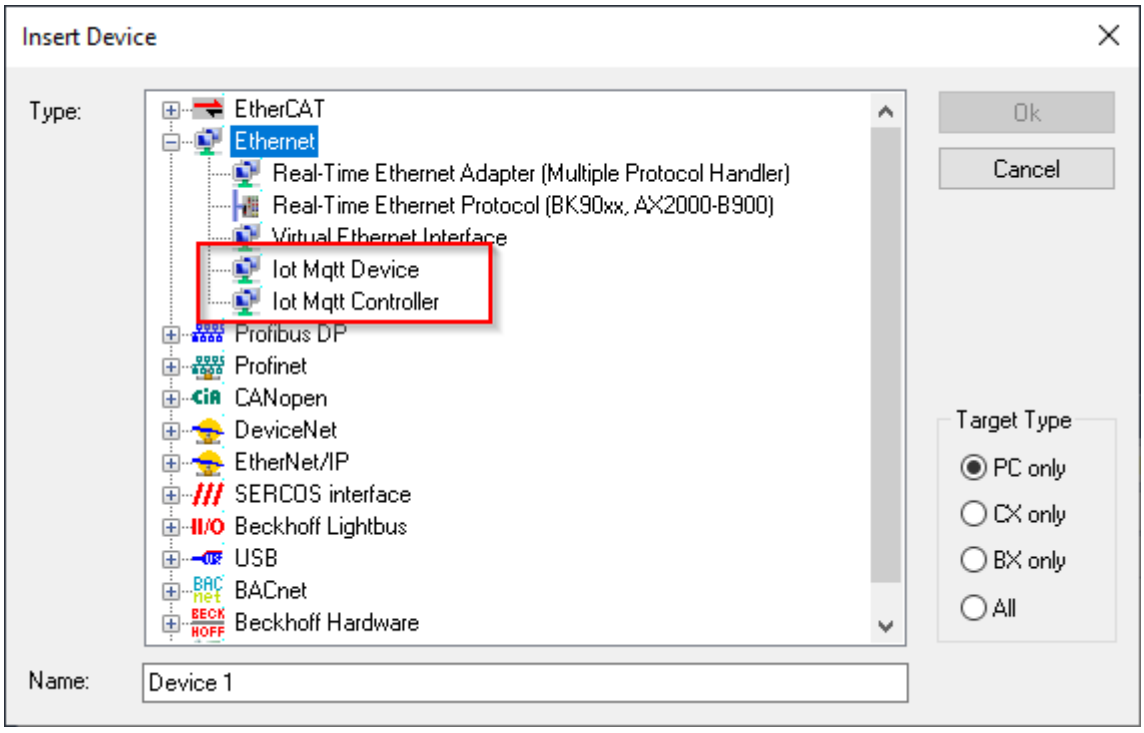
The call of the Execute method is negated by the trigger variable bReset, so that the parameters for the client ID and the host name can be reset afterwards. Once the trigger variable is set to FALSE again, the execute method is called cyclically again.

4.8 I/O device

In addition to the Tc3_IotBase 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 connection between two TwinCAT systems. Alternatively, an EK9160 IoT coupler can also be integrated into this type of communication.



The two device types can be added via the corresponding dialog in the TwinCAT I/O area and are located there within the Ethernet category.



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 as a Retain [▶ 32] message.

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.

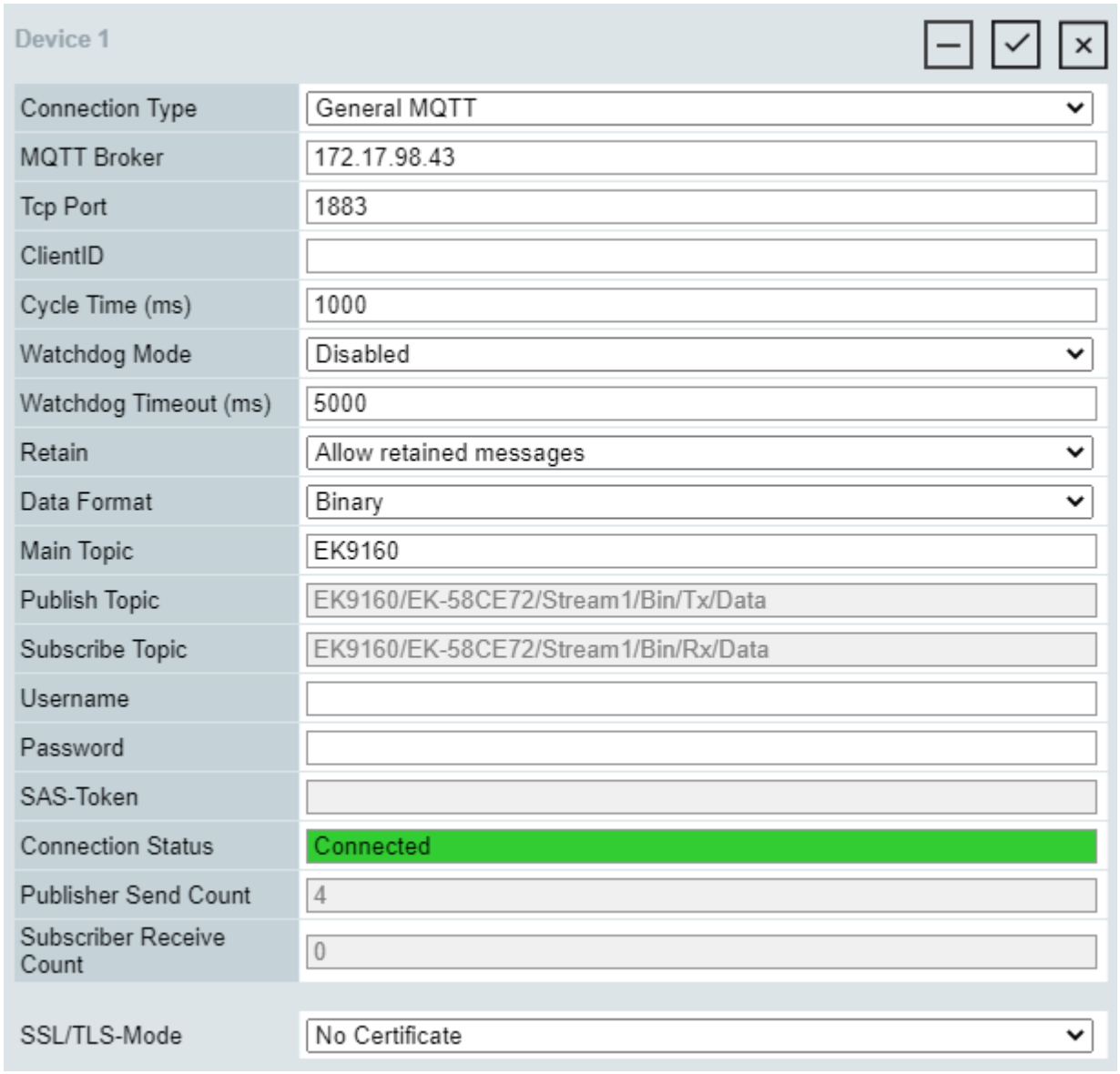
Data format

The data format for this type of MQTT communication must be set to "Binary".

EK9160 configuration

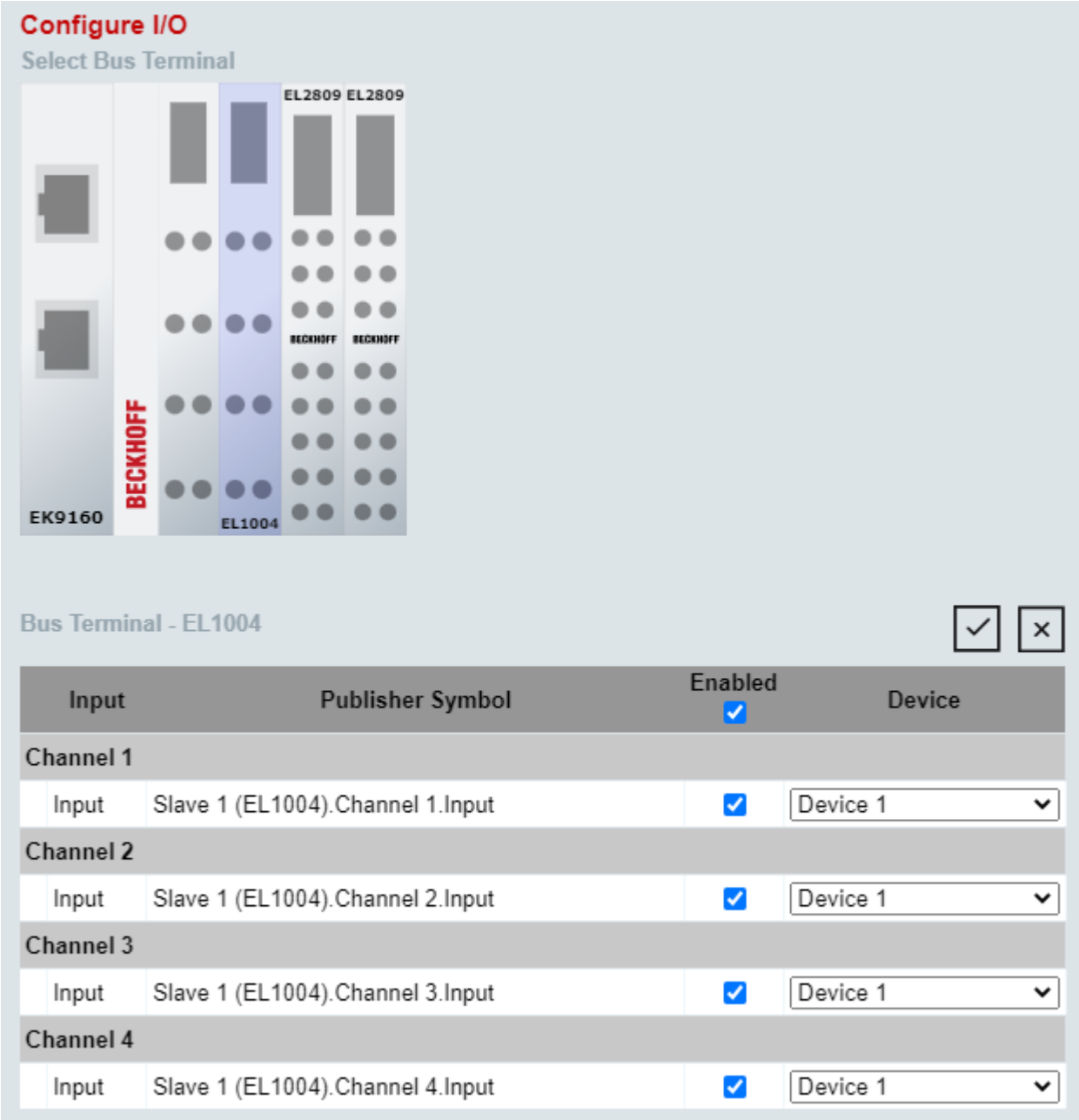
The EK9160 is automatically configured as an IoT MQTT Device in the background as soon as the device is configured to connect to a message broker. As a prerequisite, "Binary" must be selected as the data format and retain messages must be activated.

The following screenshot shows the corresponding section of the EK9160 configuration web page.



All I/O terminals have been activated for the communication connection with the message broker.

The following screenshot shows an example of this process on the configuration interface.

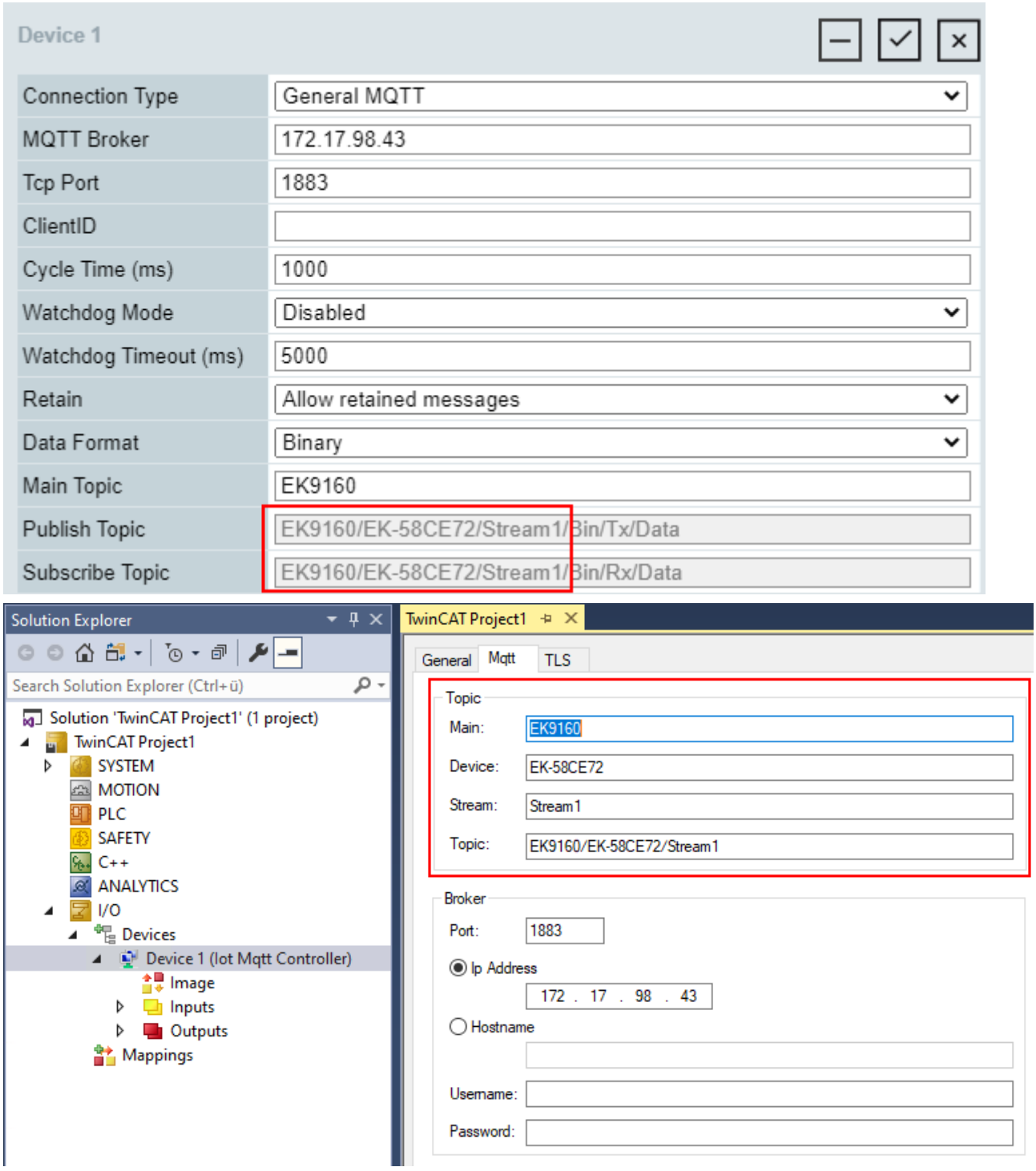


TwinCAT as IoT MQTT Device

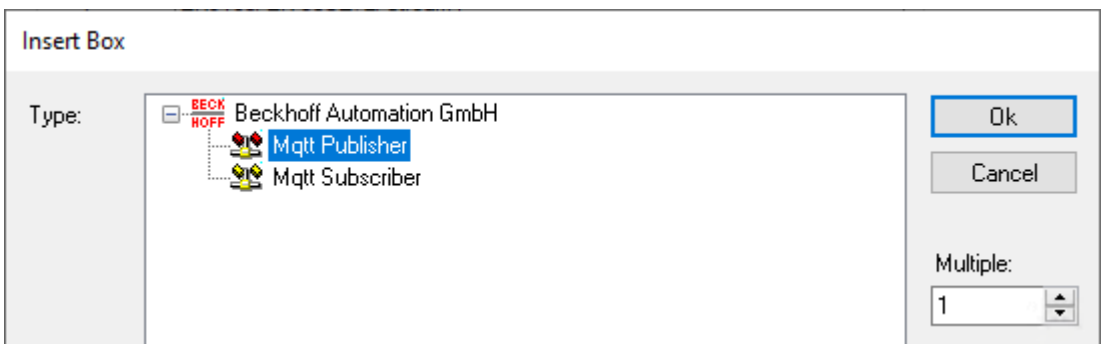
In addition to the EK9160, a TwinCAT system can also be configured as an IoT MQTT Device. In this case, the corresponding configuration steps must be carried out manually via the TwinCAT I/O area. The IoT MQTT Device then behaves in the same way as the EK9160.

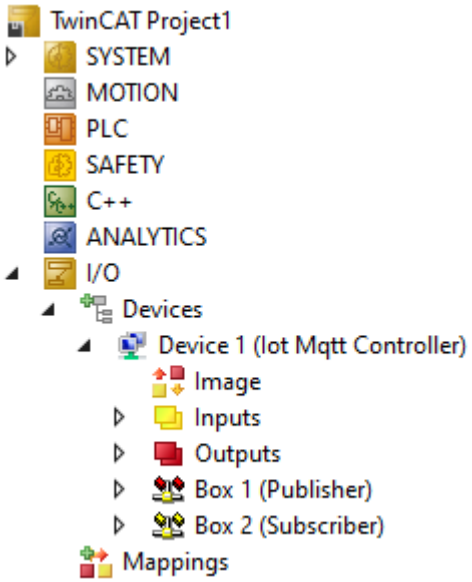
Configuration in TwinCAT

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 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.

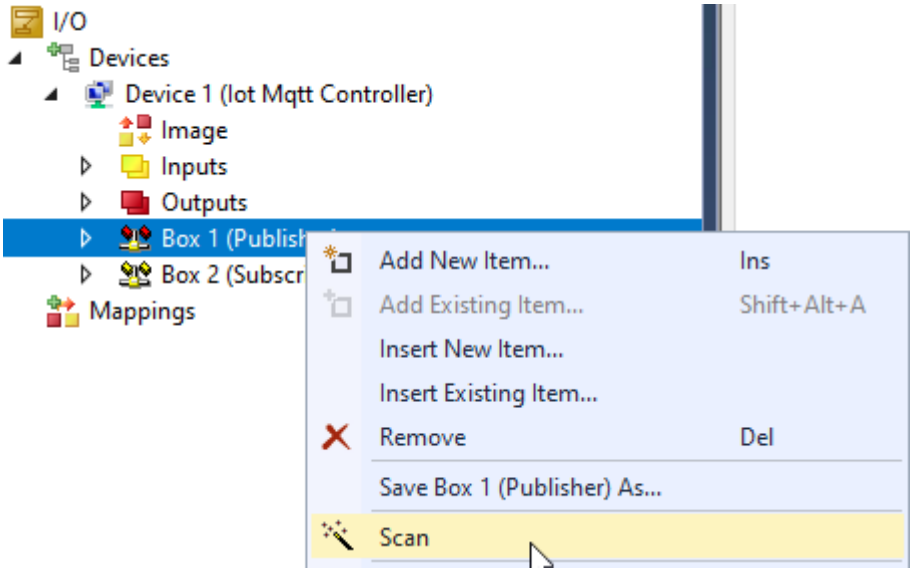


Publishers and Subscribers can then be created below the IoT MQTT controller, depending on whether you want to scan the output or input terminals. Input terminals are operated via the Publisher and output terminals via the Subscriber.

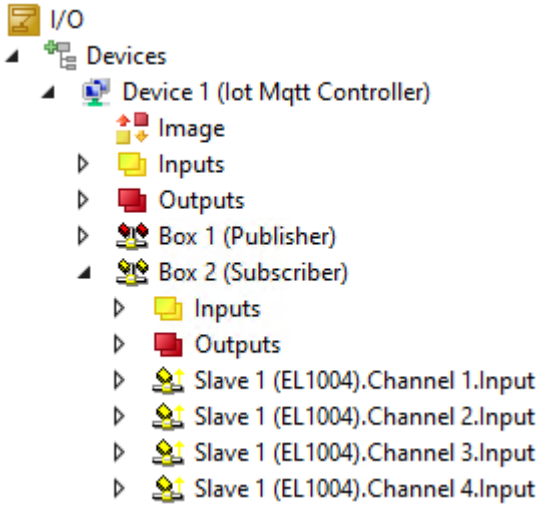




Subsequently, the symbol information can be read out via a scan mechanism and corresponding input/output variables can be automatically created in the process image of the device.



Result (taking the subscriber as an example):

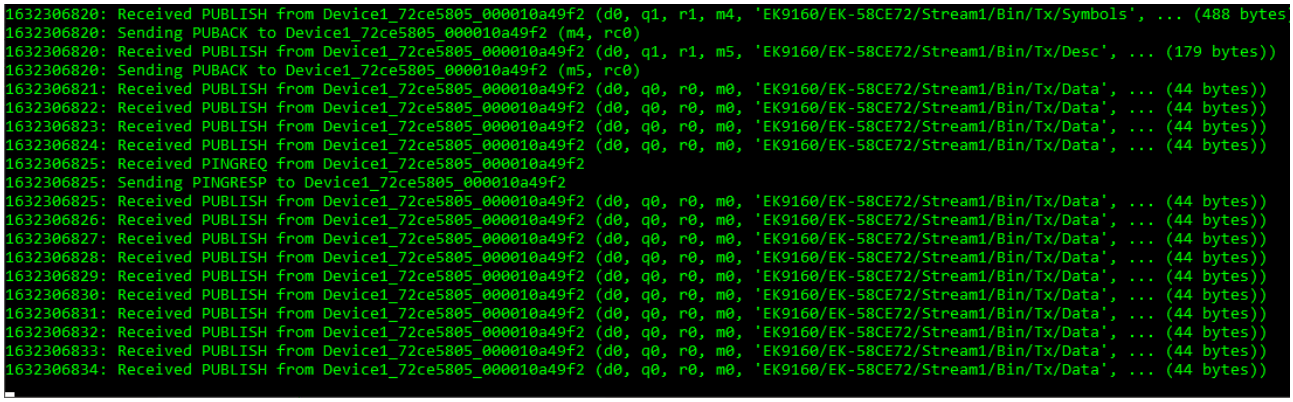


Further Information

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 connection 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.



4.9 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() [▶ 61] 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() [▶ 62] method can be used to deactivate this function programmatically.

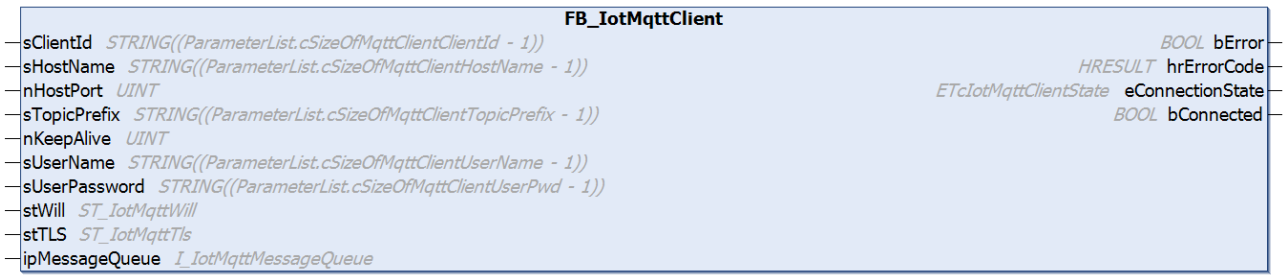
5 PLC API

5.1 Tc3_IotBase

5.1.1 MQTT3

The following function blocks and structures are implemented for MQTT in version 3.1.1. The implementations for MQTT in version 5 can be found at MQTT5 [▶ 69].

5.1.1.1 FB_IotMqttClient



The function block enables communication to a message broker based on MQTT version 3.1.1.

A client function block is responsible for the connection to precisely one broker. The Execute() [▶ 57] 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 inte
rnally)
    nKeepAlive       : UINT := 60;      // in seconds

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

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

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_IotMqttWill [▶ 62]	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_IotMqttTls [▶ 63]	If the broker offers a TLS-secured connection, the required configuration can be implemented here.
ipMessageQueue	I_IotMqttMessageQueue	An instance of FB_IotMqttMessageQueue [▶ 64] can optionally be assigned here. This input parameter can be used for storing incoming new messages in the message queue without implementing the callback method. (See also IotMqttSampleUsingQueue [▶ 312]) If you want to evaluate incoming new messages without a message queue by implementing the callback method yourself, this input parameter is not needed. (See also IotMqttSampleUsingCallback [▶ 314]) No message queue is used if the callback method is used or overwritten, irrespective of whether ipMessageQueue was set.

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	ETcIotMqttClientState [▶ 109]	Indicates the state of the connection between client and broker as enumeration ETcIotMqttClientState.
bConnected	BOOL	This output is TRUE if a connection exists between client and broker.

Methods

Name	Description
Execute [▶ 57]	Method for background communication with the TwinCAT driver. The method must be called cyclically.
Publish [▶ 58]	Method for executing a publish operation to a MQTT message broker.
Subscribe [▶ 59]	Method for establishing a subscription.
Unsubscribe [▶ 60]	Method for removing a subscription.
ActivateExponentialBackoff [▶ 61]	Activates the exponential backoff [▶ 54] function
DeactivateExponentialBackoff [▶ 62]	Deactivates the exponential backoff [▶ 54] function
GetTimeSinceLastBrokerMessage [▶ 62]	Method for querying the time (in ms) since the last message from the message broker.

Event-driven methods (callback methods)

Name	Description
OnMqttMessage [▶ 60]	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_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.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.0	IPC or CX (x86, x64, ARM)	Tc3_IotBase

5.1.1.1.1 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.

5.1.1.1.2 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 subscri
bers
    bQueue       : BOOL; // for future extension
END_VAR
```

Return value

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

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.

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

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

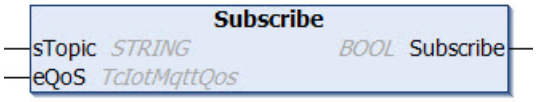
The variables of type STRING used here are based on the UTF-8 format. This STRING formatting is common for MQTT communication as well as for JSON documents.

In order to be able to receive special characters and texts from a wide range of languages, the character set in the Tc3_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.

5.1.1.1.3 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 subscribe an MQTT client instance to 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
```

Return value

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

Inputs

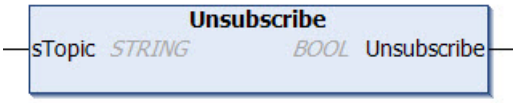
Name	Type	Description
eQoS	TcIotMqttQos	"Quality of Service"

Inputs/outputs

Name	Type	Description
sTopic	STRING	Topic of the MQTT message

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

5.1.1.1.4 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
```

Return value

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

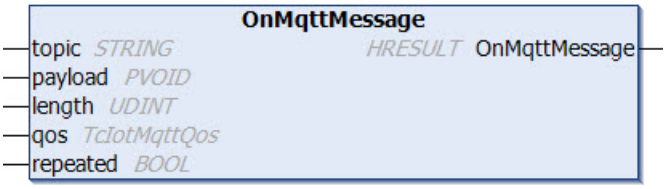
Inputs/outputs

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

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

5.1.1.1.5 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 use of the callback method is also explained in the sample IotMqttSampleUsingCallback [▶ 314].

Syntax

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

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.

Inputs

Name	Type	Description
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.

Inputs/outputs

Name	Type	Description
Topic	STRING	Topic of the received MQTT message

5.1.1.1.6 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() [▶ 61] 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() [▶ 62] method can be used to deactivate this function programmatically.

Syntax

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

Inputs

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

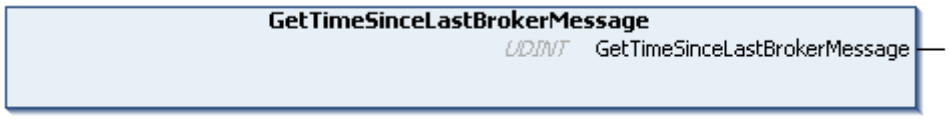
5.1.1.1.7 DeactivateExponentialBackoff

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() [▶ 61] 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() [▶ 62] method can be used to deactivate this function programmatically.

Syntax

```
METHOD DeactivateExponentialBackoff
```

5.1.1.1.8 GetTimeSinceLastBrokerMessage



This method specifies the time (in ms) since the last message from the message broker. The time is reset to 0 with every new message from the message broker. It does not matter whether these are ping messages or regular publish/subscribe messages.

The MQTT specification defines that an MQTT client itself determines the time after which it will accept a timeout in the direction of the message broker. This is possible with this method from the PLC. The keep alive time, on the other hand, specifies the time (multiplied by 1.5) after which the message broker assumes a client time-out if there are no messages from the client.

Syntax

```
METHOD GetTimeSinceLastBrokerMessage : UDINT
```

Return value

Name	Type	Description
GetTimeSinceLastBrokerMessage	UDINT	The time in milliseconds since the last message from the message broker.

5.1.1.2 ST_IotMqttWill

By means of the following specifications, a message can be specified which, in case of an irregular disconnection between client and broker, is sent as the last message from the broker to the subscribers who have subscribed to the corresponding topic.

Syntax

Definition:

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

```
ibers
END_STRUCT
END_TYPE
```

Parameter

Name	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)
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.

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

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

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

Parameters

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. If communication is to take place without TLS encryption, this value must remain FALSE.
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

5.1.1.4 FB_IotMqttMessageQueue

This function block provides a message queue for MQTT messages, which can be used with the FB_IotMqttClient [▶ 55] function block. 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). It is possible that multiple MQTT messages are received within one PLC cycle and made available at the message queue.

During the program sequence, the property `nQueuedMessages` can be used to check whether and how many messages have been collected in the message queue. The `Dequeue()` method is used for removing messages from the FIFO queue. The oldest message is output first.

Size of the MQTT message queue

The maximum number of possible messages in the queue can be set via the parameter `cMaxEntriesInMqttMessageQueue` in the parameter list [▶ 111] of the Tc3_IotBase library. 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 [▶ 111].

Properties

Name	Type	Access	Description
bOverwriteOldestEntry	BOOL	Get, Set	Here you can parameterize whether, when the queue is full, a newly received message should overwrite the oldest message. If yes (TRUE), the oldest message is lost. If no (FALSE), the latest message is lost. A full queue is an unlikely case if the user ensures a speedy readout of the queue using the `Dequeue()` method.
nQueuedMessages	UDINT	Get	Outputs the number of MQTT messages currently collected in the queue.

Methods

Name	Description
Dequeue() [▶ 65]	Removes an MQTT message from the queue.
ResetQueue() [▶ 66]	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_IotBase

5.1.1.4.1 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
```

Return value

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

Inputs

Name	Type	Description
fbMessage	REFERENCE TO FB_IotMqttMessage	For the message itself the reference is transferred to an instance of type FB_IotMqttMessage [▶ 66].

5.1.1.4.2 ResetQueue

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

Syntax

```
METHOD ResetQueue : BOOL
```

Return value

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

5.1.1.5 FB_IotMqttMessage

If the TwintCAT MQTT client (FB_IotMqttClient [▶ 55]) is used in combination with a message queue (FB_IotMqttMessageQueue [▶ 64]), 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 possible size of a message can be set using the `cMaxSizeOfMqttMessage` parameter in the parameter list [▶ 111] of the Tc3_IotBase library. By default, the message size is limited to 100 kB.

When using the MQTT Message Queue, it dynamically allocates new memory from the TwinCAT Router memory for new messages.

Properties

Name	Type	Access	Description
eQoS	TcIotMqttQos	Get	"Quality of Service" of the MQTT message
nPayloadSize	UDINT	Get	Size of the payload in bytes
nTopicSize	UINT	Get	Size of the topic in bytes

Methods

Name	Description
CompareTopic() [▶ 67]	Compares a specified topic with the topic in the MQTT message
GetTopic() [▶ 68]	Returns the topic of an MQTT message
GetPayload() [▶ 68]	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_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.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.0	IPC or CX (x86, x64, ARM)	Tc3_IotBase

5.1.1.5.1 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
```

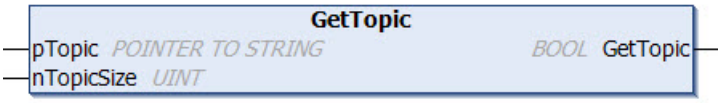
Return value

Name	Type	Description
CompareTopic	BOOL	Is TRUE, if the specified topic is identical to the topic of the MQTT message in the function block.

Inputs/outputs

Name	Type	Description
sTopic	STRING	

5.1.1.5.2 GetTopic



Syntax

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

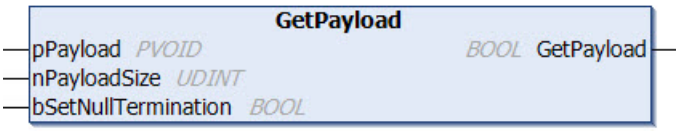
Return value

Name	Type	Description
GetTopic	BOOL	

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.

5.1.1.5.3 GetPayload



Syntax

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

Return value

Name	Type	Description
GetPayload	BOOL	

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 null termination (string), this can be implemented during the copy process. This is not necessary if the message source (publisher) has already implemented a null termination and this was taken into account in the payload size specification. In many cases no reliable information is available.

5.1.2 MQTT5

The following function blocks and structures are implemented for MQTT in version 5.

5.1.2.1 FB_IotMqtt5Client

The function block enables communication to a message broker based on MQTT version 5.0.

A client function block is responsible for the connection to precisely one broker. The Execute() 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.

Use the `FB_IotMqtt5Client`, if you do not want to implement callback methods yourself and incoming new messages should be provided in the message queue available at the output. Use the FB_IotMqtt5ClientBase if you want to implement the callback methods (`OnMqtt5Message()` and others) yourself.

Syntax

Definition:

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

    {attribute 'TcEncoding':='UTF-8'}
    sUserName        : STRING(255);        // optional parameter
    {attribute 'TcEncoding':='UTF-8'}
    sUserPassword    : STRING(255);        // optional parameter
    stWill           : ST_IotMqtt5Will;    // optional parameter
    stTLS            : ST_IotMqtt5Tls;     // optional parameter
    stAuth           : ST_IotMqtt5Auth;    // optional parameter
    stConnect        : ST_IotMqtt5Connect; // optional parameter
END_VAR
VAR_OUTPUT
    bError            : BOOL;
    hrErrorCode       : HRESULT;
    eConnectionState  : ETcIotMqttClientState;
    bConnected        : BOOL; // TRUE if connection to host is established
    fbMessageQueue    : FB_IotMqtt5MessageQueue;         // received messages are queued during call
of Execute()
    fbConnAckProps    : FB_IotMqtt5ConnAckProperties;    // info is set when a connection is
acknowledged
    fbDisconnectProps : FB_IotMqtt5DisconnectProperties; // info is set after disconnection
END_VAR
```

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_IotMqtt5Will [▶ 91]	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_IotMqtt5Tls [▶ 92]	If the broker offers a TLS-secured connection, the required configuration can be implemented here.
stAuth	ST_IotMqtt5Auth [▶ 93]	This structure can be used to pass extended authentication information to the message broker.
stConnect	ST_IotMqtt5Connect [▶ 93]	This structure can be used to pass advanced connection settings to the message broker.

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	ETcIotMqttClientState [▶ 109]	Indicates the state of the connection between client and broker as enumeration ETcIotMqttClientState.
bConnected	BOOL	This output is TRUE if a connection exists between client and broker.
fbMessageQueue	FB_IotMqtt5MessageQueue [▶ 94]	The message queue of the received messages is provided at the output. This output can be accessed directly to work with the message queue.
fbConnAckProps	FB_IotMqtt5ConnAckProperties [▶ 100]	If the client has received ConnAck Properties from the message broker, they are displayed here.
fbDisconnectProps	FB_IotMqtt5DisconnectProperties [▶ 103]	If the client has received Disconnect Properties from the message broker, they are displayed here.

Methods

Name	Description
Execute [▶ 71]	Method for background communication with the TwinCAT driver. The method must be called cyclically.
Publish [▶ 72]	Method for executing a publish operation to a MQTT message broker.
Request [▶ 76]	This can be used to send a request as part of the request/response procedure of MQTTv5.
Response [▶ 77]	This can be used to send a response as part of the request/response procedure of MQTTv5.
Subscribe [▶ 73]	Method for establishing a subscription.
Unsubscribe [▶ 74]	Method for removing a subscription.
ActivateExponentialBackoff [▶ 75]	Activates the exponential backoff [▶ 54] function
DeactivateExponentialBackoff [▶ 75]	Deactivates the exponential backoff [▶ 54] function
GetTimeSinceLastBrokerMessage [▶ 78]	Method for querying the time (in ms) since the last message from the message broker.

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_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.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4026.0	IPC or CX (x86, x64, ARM)	Tc3_IotBase (>= v3.4.2.0)

5.1.2.1.1 Execute

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

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

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.

5.1.2.1.2 Publish

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

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

Syntax

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

Return value

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

Inputs

Name	Type	Description
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.
pProps	POINTER TO MqttPublishProperties	Pointer to Publish properties to be sent. The parameter is optional. The FB_IotMqtt5PublishProperties [▶ 104] can be used to specify Publish properties.

Inputs/outputs

Name	Type	Description
sTopic	STRING	Topic of the MQTT message

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

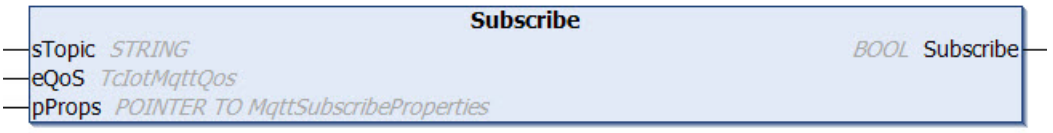
The variables of type STRING used here are based on the UTF-8 format. This STRING formatting is common for MQTT communication as well as for JSON documents.

In order to be able to receive special characters and texts from a wide range of languages, the character set in the Tc3_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.

5.1.2.1.3 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 subscribe an MQTT client instance to multiple topics.

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

Syntax

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

Return value

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

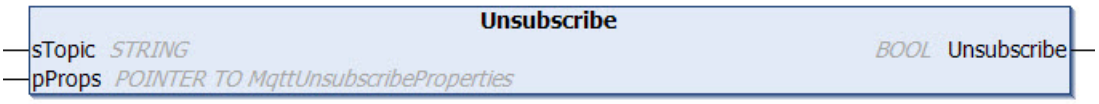
Inputs

Name	Type	Description
eQoS	TcIotMqttQos	"Quality of Service"
pProps	POINTER TO MqttSubscribeProperties	Pointer to Subscribe properties to be sent. The parameter is optional. The FB_IotMqtt5SubscribeProperties [▶ 106] can be used to specify Subscribe properties.

Inputs/outputs

Name	Type	Description
sTopic	STRING	Topic of the MQTT message

5.1.2.1.4 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.

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

Syntax

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

Return value

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

Inputs

Name	Type	Description
pProps	POINTER TO MqttUnsubscribeProperties	Pointer to Unsubscribe properties to be sent. The parameter is optional. The FB_IotMqtt5UnsubscribeProperties [▶ 107] can be used to specify Unsubscribe properties.

Inputs/outputs

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

5.1.2.1.5 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() [▶ 61] 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() [▶ 62] 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.

5.1.2.1.6 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() [▶ 61] 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() [▶ 62] method can be used to deactivate this function programmatically.

Syntax

```
METHOD DeactivateExponentialBackoff
```

5.1.2.1.7 Request

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

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

Syntax

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

Return value

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

Inputs

Name	Type	Description
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.
pProps	POINTER TO MqttPublishProperties	Pointer to Publish properties to be sent. The parameter is optional. The FB_IotMqtt5PublishProperties [▶ 104] can be used to specify Publish properties.
pCorrelationData	POINTER TO BYTE	Pointer to the CorrelationData to be sent. This can be used to assign a request to a received response in a request/response procedure.
nCorrelationDataSize	UINT	Size of the CorrelationData in bytes

Inputs/outputs

Name	Type	Description
sTopic	STRING	Topic of the MQTT message
sResponseTopic	STRING	Response topic on which the request should be answered.

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_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.

5.1.2.1.8 Response

This method is called once to send a response to a request.

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

Syntax

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

Return value

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

Inputs

Name	Type	Description
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.
pProps	POINTER TO MqttPublishProperties	Pointer to Publish properties to be sent. The parameter is optional. The FB_IotMqtt5PublishProperties [▶ 104] can be used to specify Publish properties.
pCorrelationData	POINTER TO BYTE	Pointer to the CorrelationData to be sent. This can be used to assign a request to a received response in a request/response procedure.
nCorrelationDataSize	UINT	Size of the CorrelationData in bytes

Inputs/outputs

Name	Type	Description
sResponseTopic	STRING	Topic on which the response should be sent.

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

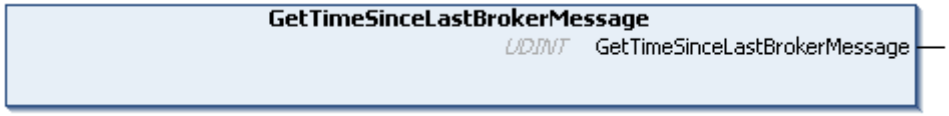
The variables of type STRING used here are based on the UTF-8 format. This STRING formatting is common for MQTT communication as well as for JSON documents.

In order to be able to receive special characters and texts from a wide range of languages, the character set in the Tc3_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.

5.1.2.1.9 GetTimeSinceLastBrokerMessage



This method specifies the time (in ms) since the last message from the message broker. The time is reset to 0 with every new message from the message broker. It does not matter whether these are ping messages or regular publish/subscribe messages.

The MQTT specification defines that an MQTT client itself determines the time after which it will accept a timeout in the direction of the message broker. This is possible with this method from the PLC. The keep alive time, on the other hand, specifies the time (multiplied by 1.5) after which the message broker assumes a client time-out if there are no messages from the client.

Syntax

```
METHOD GetTimeSinceLastBrokerMessage : UDINT
```

Return value

Name	Type	Description
GetTimeSinceLastBrokerMessage	UDINT	The time in milliseconds since the last message from the message broker.

5.1.2.2 FB_IotMqtt5ClientBase

The function block enables communication to a message broker based on MQTT version 5.0.

A client function block is responsible for the connection to precisely one broker. The Execute() 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.

Use the `FB_IotMqtt5ClientBase` if you want to implement the callback methods (`OnMqtt5Message()` and others) yourself. Use the FB_IotMqtt5Client if you do not want to implement callback methods yourself and incoming new messages should be provided in the message queue available at the output.

Syntax

Definition:

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

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

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_IotMqtt5Will [▶ 91]	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_IotMqtt5Tls [▶ 92]	If the broker offers a TLS-secured connection, the required configuration can be implemented here.
stAuth	ST_IotMqtt5Auth [▶ 93]	This structure can be used to pass extended authentication information to the message broker.
stConnect	ST_IotMqtt5Connect [▶ 93]	This structure can be used to pass advanced connection settings to the message broker.

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	ETcIotMqttClientState [▶ 109]	Indicates the state of the connection between client and broker as enumeration ETcIotMqttClientState.
bConnected	BOOL	This output is TRUE if a connection exists between client and broker.

Methods

Name	Description
Execute [▶ 82]	Method for background communication with the TwinCAT driver. The method must be called cyclically.
Publish [▶ 82]	Method for executing a publish operation to a MQTT message broker.
Request [▶ 88]	This can be used to send a request as part of the request/response procedure of MQTTv5.
Response [▶ 89]	This can be used to send a response as part of the request/response procedure of MQTTv5.
Subscribe [▶ 83]	Method for establishing a subscription.
Unsubscribe [▶ 84]	Method for removing a subscription.
ActivateExponentialBackoff [▶ 87]	Activates the exponential backoff [▶ 54] function
DeactivateExponentialBackoff [▶ 88]	Deactivates the exponential backoff [▶ 54] function
GetTimeSinceLastBrokerMessage [▶ 91]	Method for querying the time (in ms) since the last message from the message broker.

Event-driven methods (callback methods)

Name	Description
OnMqtt5Authentication	Callback method called by the TwinCAT driver when the message broker sends an AUTH message to the client in response to a Connect.
OnMqtt5ConnAck	Callback method called by the TwinCAT driver when the message broker sends a CONNACK message in response to a Connect.
OnMqtt5Disconnected	Callback method that is called by the TwinCAT driver in case of the interruption of a connection to a message broker.
OnMqtt5Message	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_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.

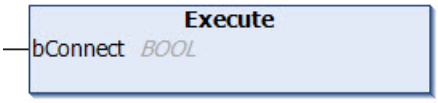
If the ASCII character set is used, there is no difference between the typical formatting of a STRING and the UTF-8 formatting of a STRING.

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.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4026.0	IPC or CX (x86, x64, ARM)	Tc3_IotBase (>= v3.4.2.0)

5.1.2.2.1 Execute



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

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

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.

5.1.2.2.2 Publish

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

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

Syntax

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

Return value

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

Inputs

Name	Type	Description
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.
pProps	POINTER TO MqttPublishProperties	Pointer to Publish properties to be sent. The parameter is optional. The FB_IotMqtt5PublishProperties [▶ 104] can be used to specify Publish properties.

Inputs/outputs

Name	Type	Description
sTopic	STRING	Topic of the MQTT message

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

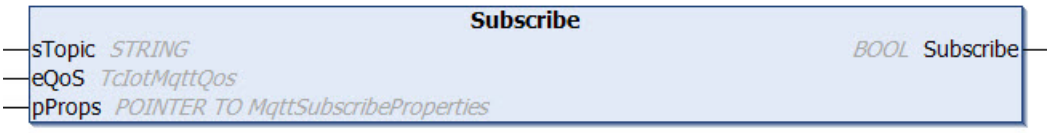
The variables of type STRING used here are based on the UTF-8 format. This STRING formatting is common for MQTT communication as well as for JSON documents.

In order to be able to receive special characters and texts from a wide range of languages, the character set in the Tc3_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.

5.1.2.2.3 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 subscribe an MQTT client instance to multiple topics.

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

Syntax

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

Return value

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

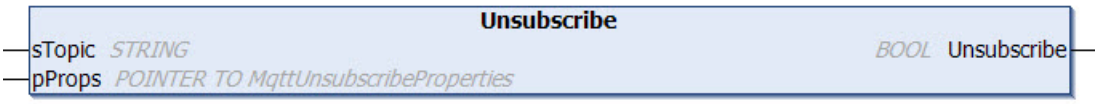
Inputs

Name	Type	Description
eQoS	TcIotMqttQos	"Quality of Service"
pProps	POINTER TO MqttSubscribeProperties	Pointer to Subscribe properties to be sent. The parameter is optional. The FB_IotMqtt5SubscribeProperties [▶ 106] can be used to specify Subscribe properties.

Inputs/outputs

Name	Type	Description
sTopic	STRING	Topic of the MQTT message

5.1.2.2.4 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.

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

Syntax

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

Return value

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

Inputs

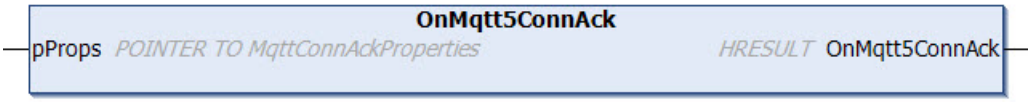
Name	Type	Description
pProps	POINTER TO MqttUnsubscribeProperties	Pointer to Unsubscribe properties to be sent. The parameter is optional. The FB_IotMqtt5UnsubscribeProperties [▶ 107] can be used to specify Unsubscribe properties.

Inputs/outputs

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

5.1.2.2.5 OnMqtt5ConnAck

Callback method



This method must not be called by the user. Instead, the function block FB_IotMqtt5ClientBase can be used to derive information and overwrite this method. During the call of the Execute() method the responsible TwinCAT driver has the possibility to receive a notification via this callback, if the message broker has sent a CONNACK as response to a Connect.

Syntax

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

Return value

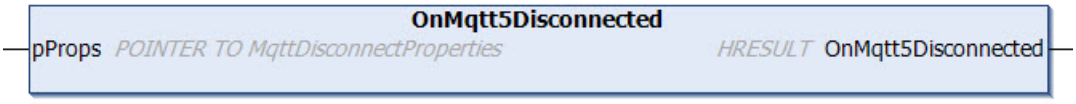
Name	Type	Description
OnMqtt5ConnAck	HRESULT	

Inputs

Name	Type	Description
pProps	POINTER TO MqttConnAck Properties	MQTT properties received in case of CONNACK.

5.1.2.2.6 OnMqtt5Disconnected

Callback method



This method must not be called by the user. Instead, the function block FB_IotMqtt5ClientBase can be used to derive information and overwrite this method. During the call of the Execute() method the responsible TwinCAT driver has the possibility to call the OnMqtt5Disconnected() method in case of a disconnection with the message broker.

Syntax

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

Return value

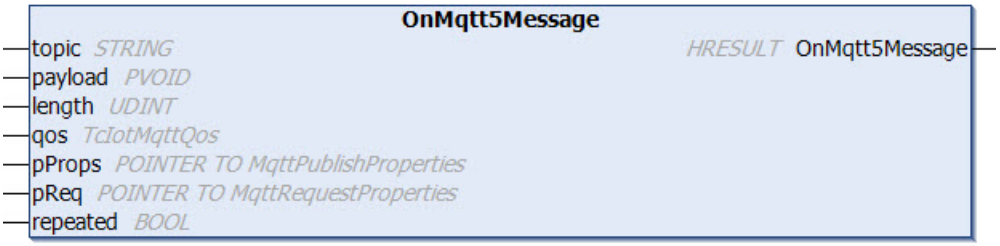
Name	Type	Description
OnMqtt5Disconnect	HRESULT	

Inputs

Name	Type	Description
pProps	POINTER TO MqttDisconnectProperties	MQTT properties received in case of Disconnect.

5.1.2.2.7 OnMqtt5Message

Callback method



This method must not be called by the user. Instead, the function block FB_IotMqtt5ClientBase can be used to derive information and overwrite this method. While the Execute() method is called, the responsible TwinCAT driver can call the OnMqtt5Message() 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 use of the callback method is also explained in the sample IotMqttSampleUsingCallback [▶ 314].

Syntax

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

Return value

Name	Type	Description
OnMqtt5Message	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.

Inputs

Name	Type	Description
payload	PVOID	Address for the payload of the received MQTT message
length	UDINT	Size of the payload in bytes
qos	TcIotMqttQos	"Quality of Service"
pProps	POINTER TO MqttPublishProperties	MQTT properties that may have been received with a message.
pReq	POINTER TO MqttRequestProperties	MQTT Request Properties, which may have been received with a message.
repeated	BOOL	If the user did not respond with S_OK to the last OnMqtt5Message() 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.

Inputs/outputs

Name	Type	Description
Topic	STRING	Topic of the received MQTT message

5.1.2.2.8 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() [▶ 61] 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() [▶ 62] 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.

5.1.2.2.9 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() [▶ 61] 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() [▶ 62] method can be used to deactivate this function programmatically.

Syntax

```
METHOD DeactivateExponentialBackoff
```

5.1.2.2.10 Request

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

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

Syntax

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

Return value

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

Inputs

Name	Type	Description
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.
pProps	POINTER TO MqttPublishProperties	Pointer to Publish properties to be sent. The parameter is optional. The FB_IotMqtt5PublishProperties [▶ 104] can be used to specify Publish properties.
pCorrelationData	POINTER TO BYTE	Pointer to the CorrelationData to be sent. This can be used to assign a request to a received response in a request/response procedure.
nCorrelationDataSize	UINT	Size of the CorrelationData in bytes

Inputs/outputs

Name	Type	Description
sTopic	STRING	Topic of the MQTT message
sResponseTopic	STRING	Response topic on which the request should be answered.

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_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.

5.1.2.2.11 Response

This method is called once to send a response to a request.

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

Syntax

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

Return value

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

Inputs

Name	Type	Description
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.
pProps	POINTER TO MqttPublishProperties	Pointer to Publish properties to be sent. The parameter is optional. The FB_IotMqtt5PublishProperties [▶ 104] can be used to specify Publish properties.
pCorrelationData	POINTER TO BYTE	Pointer to the CorrelationData to be sent. This can be used to assign a request to a received response in a request/response procedure.
nCorrelationDataSize	UINT	Size of the CorrelationData in bytes

Inputs/outputs

Name	Type	Description
sResponseTopic	STRING	Topic on which the response should be sent.

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

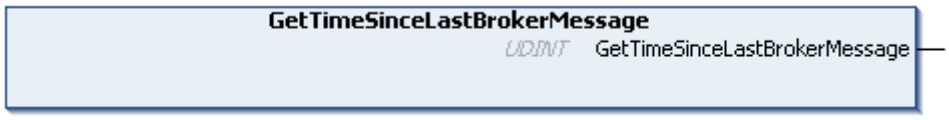
The variables of type STRING used here are based on the UTF-8 format. This STRING formatting is common for MQTT communication as well as for JSON documents.

In order to be able to receive special characters and texts from a wide range of languages, the character set in the Tc3_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.

5.1.2.2.12 GetTimeSinceLastBrokerMessage



This method specifies the time (in ms) since the last message from the message broker. The time is reset to 0 with every new message from the message broker. It does not matter whether these are ping messages or regular publish/subscribe messages.

The MQTT specification defines that an MQTT client itself determines the time after which it will accept a timeout in the direction of the message broker. This is possible with this method from the PLC. The keep alive time, on the other hand, specifies the time (multiplied by 1.5) after which the message broker assumes a client time-out if there are no messages from the client.

Syntax

```
METHOD GetTimeSinceLastBrokerMessage : UDINT
```

Return value

Name	Type	Description
GetTimeSinceLastBrokerMessage	UDINT	The time in milliseconds since the last message from the message broker.

5.1.2.3 ST_IotMqtt5Will

By means of the following specifications, a message can be specified which, in case of an irregular disconnection between client and broker, is sent as the last message from the broker to the subscribers who have subscribed to the corresponding topic. This structure is an extension of the previous structure ST_IotMqttWill [▶ 62] for MQTT V5.

No instantiation and assignment of this structure

This structure does not allow instantiation and assignment to FB_IotMqtt5Client or FB_IotMqtt5ClientBase. Instead, the input parameter of the MQTT v5 client function block is used directly.

Syntax

Definition:

```
TYPE ST_IotMqtt5Will :
STRUCT
    {attribute 'TcEncoding':='UTF-8'}
    sTopic              : STRING(255);
```

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

Parameters

Name	Type	Description
sTopic	STRING(255)	Message topic.
fbPayload	FB_IotDataBuffer	Data buffer for the payload of the message.
eQoS	TcIotMqttQos	The "Quality of Service" offers the following setting options: QoS level 0, QoS level 1, QoS level 2 (See Neuer Knoten).
bRetain	BOOL	If bRetain is TRUE, the broker stores the message in order to make it available to subsequent subscribers.
sContentType	STRING(255)	Describes the encoding of the payload. MIME types such as "text/plain" are particularly suitable for high compatibility.
sResponseTopic	STRING(255)	At this point, the response topic can be specified on which a response from another client is expected.
nMsgExpiryInterval	UDINT	Time in seconds after which a Last Will message expires and is no longer delivered by the broker if the connection is lost.
nDelay	UDINT	Delay interval in seconds until the LastWill message is sent.
bPayloadUtf8	BOOL	Format indicator for the message content.
fbCorrelationData	FB_IotDataBuffer	Data buffer for CorrelationData
fbUserProperties	FB_IotMqtt5UserProperties [▶ 108]	UserProperties can be specified at this point.

5.1.2.4 ST_IotMqtt5Tls

TLS security settings for the MQTT client (V5).

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

Syntax

Definition:

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

    sPskIdentity       : STRING(255);
```

```
    aPskKey             : ARRAY[1..64] OF BYTE;
    nPskKeyLen          : USINT;

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

Parameters

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. If communication is to take place without TLS encryption, this value must remain FALSE.
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

5.1.2.5 ST_IotMqtt5Auth

Authentication settings for the MQTT client (V5).

Syntax

Definition:

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

Parameter

Name	Type	Description
sAuthMethod	STRING(255)	UTF-8 encoded name of an extended authentication method.
aAuthData	ARRAY[0..4095] OF BYTE	This array contains binary authentication data.
nAuthDataSize	UINT	Specifies the length of the authentication data.

5.1.2.6 ST_IotMqtt5Connect

Connection settings for the MQTTv5 client, which can be set at the function block FB_IotMqtt5Client [▶ 79]. Through these settings, the client can transmit information to the message broker about potential restrictions, for example, a maximum packet size that the client can receive.

Syntax

Definition:

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

Parameter

Name	Type	Description
nSessionExpire	UDINT	Specifies the time in seconds after which a message broker deletes a client session if the client is not connected.
nMaxPacketSize	UDINT	With this setting the client tells the message broker up to which size in bytes it wants to or can receive messages. When a larger message is received, the client terminates the connection.
nReceiveMax	UINT	This setting determines how many QoS 1 or QoS 2 messages may be communicated to the client simultaneously. Simultaneous in the case means that the handshake (two- or four-step) happens at the same time.
nTopicAliasMax	UINT	Specifies the highest value a Topic Alias may have to limit the number of Topic Aliases existing at the same time. If the value is 0, it means that the client does not accept Topic Aliases.
bReqResponseInfo	BOOL	If the value is TRUE, then the message broker may send information in the CONNACK response. If the value is FALSE, the message broker does not send any response information.
bIgnoreProblemInfo	BOOL	If the value is TRUE, then the message broker may send a Reason String or User Properties with each packet. If the value is FALSE, the message broker may

5.1.2.7 FB_IotMqtt5MessageQueue

This function block provides a message queue for MQTT messages that are received with the function block FB_IotMqtt5Client [▶ 69]. For this purpose, an instance is already provided at the output of `FB_IotMqtt5Client`. The function block operates based on the first in, first out principle (FIFO). It is possible that multiple MQTT messages are received within one PLC cycle and made available at the message queue.

During the program sequence, the property `nQueuedMessages` can be used to check whether and how many messages have been collected in the message queue. The `Dequeue()` method is used for removing messages from the FIFO queue. The oldest message is output first.

Size of the MQTT message queue

The maximum number of possible messages in the queue can be set via the parameter `cMaxEntriesInMqttMessageQueue` in the parameter list [▶ 111] of the Tc3_IotBase library. 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 [▶ 111].

Properties

Name	Type	Access	Description
bOverwriteOldestEntry	BOOL	Get, Set	Here you can parameterize whether, when the queue is full, a newly received message should overwrite the oldest message. If yes (TRUE), the oldest message is lost. If no (FALSE), the reception of the new message will wait until the queue offers space. A full queue is an unlikely case if the user ensures a speedy readout of the queue using the `Dequeue()` method.
nLostMessages	UDINT	Get	Returns the number of MQTT messages which had to be discarded completely. Discarded MQTT messages can be caused by a full queue or by the total size of a newly received message being too large.
nMaxSizeOfMessage	UDINT	Get	Maximum size in bytes of all received MQTT messages, which should be received by the FB_IotMqtt5Client and stored in the FB_IotMqtt5MessageQueue. If the maximum size specified with the library parameter [▶ 111] `ParameterList.cMaxSizeOfMqttMessage` is exceeded, the message queue first attempts to save the message without user properties. If this size reduction is not sufficient, the MQTT message must be discarded completely.
nQueuedMessages	UDINT	Get	Outputs the number of MQTT messages currently collected in the queue. The maximum possible number is set with the library parameter [▶ 111] `ParameterList.cMaxEntriesInMqttMessageQueue`.

Methods

Name	Description
Dequeue() [▶ 96]	Removes an MQTT message from the queue.
ResetQueue() [▶ 96]	Deletes all messages from the queue.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4026.0	IPC or CX (x86, x64, ARM)	Tc3_IotBase (>= v3.4.2.0)

5.1.2.7.1 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_IotMqtt5Message;
END_VAR
```

Return value

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

Inputs

Name	Type	Description
fbMessage	REFERENCE TO FB_IotMqtt5Message	For the message itself the reference is transferred to an instance of type FB_IotMqtt5Message.

5.1.2.7.2 ResetQueue

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

Syntax

```
METHOD ResetQueue : BOOL
```

Return value

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

5.1.2.8 FB_IotMqtt5Message

If the TwintCAT MQTT v5 client (FB_IotMqtt5Client [▶ 69]) is used in combination with a message queue (FB_IotMqtt5MessageQueue [▶ 94]), an MQTT message is represented by the FB_IotMqtt5Message function block. 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 possible size of a message can be set using the `cMaxSizeOfMqttMessage` parameter in the parameter list [▶ 111] of the Tc3_IotBase library. By default, the message size is limited to 100 kB.

When using the MQTT Message Queue, it dynamically allocates new memory from the TwinCAT Router memory for new messages.

Properties

Name	Type	Access	Description
bPayloadUtf8	BOOL	Get	If TRUE, the payload is UTF-8 formatted text.
bTopicAlias	BOOL	Get	Indicates whether the receive topic is an alias.
eQoS	TcIotMqttQos	Get	"Quality of Service" of the MQTT message
nCorrelationDataSize	UINT	Get	Size of the Correlation Data in bytes
nMsgExpiryInterval	UDINT	Get	Specifies the expiration interval of the message in seconds.
nPayloadSize	UDINT	Get	Size of the payload in bytes
nSubIdCnt	UINT	Get	Indicates how many subscription identifiers were received.
nUserPropertyCnt	UINT	Get	Number of existing user properties. These can be read using GetUserPropertyByIdx().
nUserPropertyCntLost	UINT	Get	Number of user properties discarded on reception. The cause can be an exceeding of the defined maximum number (see library parameter [▶ 111] `ParameterList.cMaxMqtt5UserProps`) or a too large received MQTT message (see library parameter [▶ 111] `ParameterList.cMaxSizeOfMqttMessage`).
nTopicSize	UINT	Get	Size of the topic in bytes
sContentType	STRING	Get	Content Type of the MQTT message If the original text is too large for this property, the GetContentType() method can be used to get the full text.

Methods

Name	Description
CompareTopic() [▶ 98]	Compares a specified topic with the topic in the MQTT message.
GetContentType()	Returns the Content Type of the MQTT message.
GetCorrelationData()	Returns the Correlation Data of the MQTT message.
GetResponseTopic()	Returns the Response Topic.
GetPayload() [▶ 99]	Returns the content of an MQTT message.
GetSubIds()	Returns the Subscription Identifiers received for the MQTT message.
GetTopic() [▶ 99]	Returns the topic of an MQTT message.
GetUserPropertyByIdx()	Returns the name and value of a UserProperty specified by its position (index) in the list.
GetUserPropertyValueByName()	Returns the value of a UserProperty specified by its name.

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.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4026.0	IPC or CX (x86, x64, ARM)	Tc3_IotBase (>= v3.4.2.0)

5.1.2.8.1 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
```

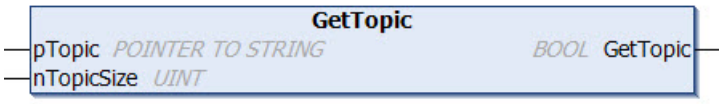
Return value

Name	Type	Description
CompareTopic	BOOL	Is TRUE, if the specified topic is identical to the topic of the MQTT message in the function block.

Inputs/outputs

Name	Type	Description
sTopic	STRING	

5.1.2.8.2 GetTopic



Syntax

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

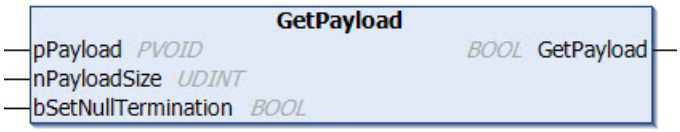
Return value

Name	Type	Description
GetTopic	BOOL	

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.

5.1.2.8.3 GetPayload



Syntax

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

Return value

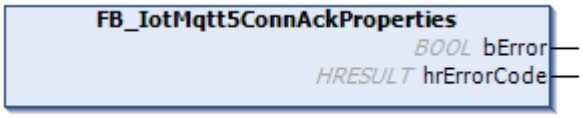
Name	Type	Description
GetPayload	BOOL	

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 null termination (string), this can be implemented during the copy process. This is not necessary if the message source (publisher) has already implemented a null termination and this was taken into account in the payload size specification. In many cases no reliable information is available.

5.1.2.9 MQTT5 Properties

5.1.2.9.1 FB_IotMqtt5ConnAckProperties



The function block enables the reception of status information from the message broker directly after the Connection Acknowledgement, which is part of the connection setup. This allows the message broker to communicate information to the client about potential constraints, such as a maximum packet size that the message broker can handle.

It is the client's task to query these values and to react to them if necessary. If the client has received ConnAck properties and makes them available at the output, this is indicated by bPropertiesAvailable=TRUE.

General description of the Connection Acknowledgement Properties:

Property	Description
Retain available	Specifies whether the message broker supports retain messages.
Server KeepAlive	Specifies the KeepAlive time assigned by the server.
Shared subscriptions available	Specifies whether the message broker supports shared subscriptions.
Wildcard subscriptions available	Specifies whether the message broker supports wildcard subscriptions, e.g. subscriptions to a # or + topic.
Authentication data	Contains the authentication data, depending on the authentication method used.
Maximum packet size	Specifies the maximum packet size of the control packet. If this value is not specified, then no maximum size is explicitly set.
Maximum QoS	Specifies the maximum QoS level that the message broker supports. For example, some message broker implementations do not support QoS 2.
Reason code	Optional reason code that the message broker can transmit to the client as part of the ConnAck procedure. In conjunction with the Server Reference (see below), the message broker can use this to inform the client, for example, that it is temporarily or permanently unavailable and/or can be reached at a new address.
Maximum receive	The server uses this property to limit the number of QoS1 and QoS2 publishes it is willing to process for the client simultaneously.
Session expiry interval	The server can use this to inform the client that it is using a different Session Expiry Interval than originally requested by the client.
Response info	Optional return information from the message broker to the client. A use case for this can be, for example, that the broker informs the client about a topic area which has been assigned to the client and which it can access.
Maximum topic alias	Specifies the maximum value that can be used for a topic alias.
Assigned client ID	Returns the ClientID assigned by the message broker if the client has not specified its own ClientID.
Server reference	Optional (address) reference to another message broker, e.g. for Reason Code 0x9C or 0x9D (Server unavailable, Server has moved).
User Properties	User Properties are key/value pairs that can be attached to the PublishProperties. This is done by means of the function block FB_IotMqtt5UserProperties [▶ 108]. The meaning of the UserProperties is not part of the MQTT5 specification and therefore application specific.

Syntax

Definition:

```
FUNCTION_BLOCK FB_IotMqtt5ConnAckProperties EXTENDS FB_IotMqtt5UserProperties [▶ 108]
VAR_OUTPUT
    bError          : BOOL;
    hrErrorCode     : HRESULT;
END_VAR
```

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.

Properties

Name	Type	Access	Description
bPropertiesAvailable	BOOL	Get	Indicates whether properties are present.
bRetainAvailable	BOOL	Get	Specifies whether the message broker supports retain messages.
bServerKeepAlive	BOOL	Get	Indicates whether a different KeepAlive value is used by the server than was originally requested by the client.
bSessionPresent	BOOL	Get	Indicates whether the client already has a session with the broker.
bSharedSubAvailable	BOOL	Get	Specifies whether the message broker supports shared subscriptions.
bSubIdAvailable	BOOL	Get	Specifies whether the message broker supports the use of Subscription Identifiers.
bWildcardSubAvailable	BOOL	Get	Specifies whether the message broker supports Wildcard Subscriptions.
nAuthDataSize	UINT	Get	Specifies the size of the Authentication Data.
nMaxPackateSize	UDINT	Get	Specifies the maximum packet size of the control packet.
nMaxQoS	BYTE	Get	Specifies the maximum QoS level that the message broker supports.
nReasonCode	BYTE	Get	Specifies an optional reason code that the server transmits to the client as part of the ConnAck procedure.
nReceiveMax	UINT	Get	Maximum number of simultaneous QoS 1 and 2 publishes that the server supports.
nResponseInfoSize	UINT	Get	Returns the size of the response information.
nSessionExpiryInterval	UDINT	Get	Value in seconds for the Session Expiry Interval supported by the server.
nTopicAliasMax	UINT	Get	Maximum value for the Topic Alias that the server supports.
sAssignedClientId	STRING	Get	Client ID assigned by the message broker.
sAuthMethod	STRING	Get	Returns the name of the authentication method used.
sReasonString	STRING	Get	Readable variant of the reason code.
sServerReference	STRING	Get	Optional Server Reference, the format is not defined by the specification.

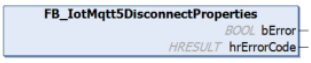
Methods

Name	Description
GetAuthData	Returns the authentication data, which depends on the used authentication method.
GetResponseInfo	Allows the response information to be read out. This allows the message broker to transmit to the client a basis for generating the response topic in the Req/Res procedure [▶ 33]. The format is not defined by the specification.
SetConnAckProperties	Enables the setting of the ConnAck properties.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4026.0	IPC or CX (x86, x64, ARM)	Tc3_IotBase (>= v3.4.2.0)

5.1.2.9.2 FB_IotMqtt5DisconnectProperties



The function block enables the reception of disconnection information.

If the client has received Disconnect Properties and provides these at the output, this is indicated by bPropertiesAvailable=TRUE.

General description of the Disconnect Properties:

Property	Description
Reason Code	Specifies the reason for the Disconnect as described in MQTTv5 specification chapter 3.14.2.1. Usually used for programmatic evaluation.
Reason String	Specifies the reason for the Disconnect in legible form. Usually not used for programmatic evaluation.
Session Expiry Interval	Indicates the expiration interval for the session (in seconds).
Server Reference	Can be used by the message broker in conjunction with ReasonCode 0x9C (Use Another Server) or 0x9D (Server moved) to specify a new server. The format of the server reference is not prescribed by the specification and therefore depends on the message broker used.
User Properties	User Properties are key/value pairs that can be attached to the PublishProperties. This is done by means of the function block FB_IotMqtt5UserProperties [▶ 108]. The meaning of the UserProperties is not part of the MQTT5 specification and therefore application specific.

Syntax

Definition:

```
FUNCTION_BLOCK FB_IotMqtt5DisconnectProperties EXTENDS FB_IotMqtt5UserProperties [▶ 108]
VAR_OUTPUT
    bError          : BOOL;
    hrErrorCode     : HRESULT;
END_VAR
```

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.

Properties

Name	Type	Access	Description
bPropertiesAvailable	BOOL	Get	Indicates whether properties are present.
nReasonCode	BYTE	Get	Specifies the Disconnect reason as a numeric value.
nSessionExpire	UDINT	Get	Specifies a lifetime in seconds for the session.
sReasonString	STRING	Get	Specifies the disconnect reason as a readable value.
sServerReference	STRING	Get	Specifies an optional server reference for ReasonCode 0x9C or 0x9D.

Methods

Name	Description
SetDisconnectProperties	Allows you to set the Disconnect properties.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4026.0	IPC or CX (x86, x64, ARM)	Tc3_IotBase (>= v3.4.2.0)

5.1.2.9.3 FB_IotMqtt5PublishProperties

The function block allows the definition of various properties that can be set when sending a message. This allows additional metadata to be attached to a message during transmission. The following metadata can be defined:

General description of the Publish Properties:

Property	Description
Content Type	Can be used by applications to convey a content description of the payload.
Topic Alias	A Topic Alias can be used to represent a topic e.g. by an integer value instead of a (potentially long) string.
Subscription Identifier	An ID to be able to identify a subscription.
Message Expiry Interval	Specifies the lifetime of a message until it can be discarded by the message broker.
Payload UTF-8 Indicator	Indicates whether the transmitted message is in UTF-8 format.
User Properties	User Properties are key/value pairs that can be attached to the PublishProperties. This is done by means of the function block FB_IotMqtt5UserProperties [▶ 108]. The meaning of the UserProperties is not part of the MQTT5 specification and therefore application specific.

Syntax

Definition:

```
FUNCTION_BLOCK FB_IotMqtt5PublishProperties EXTENDS FB_IotMqtt5UserProperties [▶ 108]
VAR_OUTPUT
    bError          : BOOL;
    hrErrorCode     : HRESULT;
END_VAR
```

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.

Properties

Name	Type	Access	Description
bPayloadUtf8	BOOL	Get, Set	Indicates whether the message content is in UTF-8 format.
bTopicAlias	BOOL	Get, Set	Specifies a Topic Alias.
nMsgExpiryInterval	UDINT	Get, Set	Specifies a lifetime in seconds for the message.
nSubIdCnt	UINT	Get	Returns the number of subscription ids.
pPublishProperties	POINTER TO MqttPublishProperties	Get	Pointer to an object of type MqttPublishProperties. When calling FB_IotMqtt5Client.Publish(), as well as Request() and Response() it is possible to pass this directly.
sContentType	STRING	Get, Set	Specifies a content type for the message content.

Methods

Name	Description
GetSubIds	Returns the subscription IDs.
SetPublishProperties	Sets the configured PublishProperties.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4026.0	IPC or CX (x86, x64, ARM)	Tc3_IotBase (>= v3.4.2.0)

5.1.2.9.4 FB_IotMqtt5SubscribeProperties

The function block allows the definition of different properties, which can be set when creating a subscription. This allows additional properties to be defined for a subscription and transferred from the client to the message broker.

General description of the Subscribe Properties:

Property	Description
No Local	When using MQTTv3, if a message is sent on the same topic that was also subscribed to, the sent message will be received again. By using this flag, you will not receive messages that have already been sent again.
Retained Message Control	Retain messages still work as with MQTTv3, but this flag adds options that define what should happen when retain messages are received.
Subscription Identifier	A numeric value used to identify a subscription.
User Properties	User Properties are key/value pairs that can transport additional metadata. These are managed via the function block FB_IotMqtt5UserProperties [▶ 108]. The meaning of the UserProperties is not part of the MQTT5 specification and therefore application specific.

Syntax

Definition:

```
FUNCTION_BLOCK FB_IotMqtt5SubscribeProperties EXTENDS FB_IotMqtt5UserProperties [▶ 108]
VAR_OUTPUT
    bError          : BOOL;
    hrErrorCode     : HRESULT;
END_VAR
```

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.

Properties

Name	Type	Access	Description
bNoLocal	BOOL	Get, Set	Specifies whether the NoLocal property should be set, see above.
bRetainAsPublished	BOOL	Get, Set	Specifies whether the retain flag of a publisher should remain set when a message is received.
nRetainHandling	BYTE	Get, Set	Specifies how to receive retain messages. 0: Retain messages are sent by the broker when the client subscribes. (Default behavior of MQTTv3) 1: Retain messages are sent by the broker only when the client subscribes if the subscription does not exist. 2: Retain messages are not sent by the broker when the client subscribes.
nSubId	UDINT	Get, Set	Numeric value for optional identification of a subscription.
pSubscribePropertie s	POINTER TO MqttSubscribeProperties	Get	Pointer to an object of type MqttSubscribeProperties. When calling FB_IotMqtt5Client.Subscribe() it is possible to pass this directly.

Methods

Name	Description
SetSubscribeProperties	Allows you to set the Subscribe properties.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4026.0	IPC or CX (x86, x64, ARM)	Tc3_IotBase (>= v3.4.2.0)

5.1.2.9.5 FB_IotMqtt5UnsubscribeProperties

The function block allows the definition of various properties that can be set when a subscription is unsubscribed. This allows the following metadata to be transmitted from the client to the message broker.

General description of the Unsubscribe Properties:

Property	Description
User Properties	User Properties are key/value pairs that can transport additional metadata. These are managed via the function block FB_IotMqtt5UserProperties [▶ 108]. The meaning of the UserProperties is not part of the MQTT5 specification and therefore application specific.

Syntax

Definition:

```
FUNCTION_BLOCK FB_IotMqtt5UnsubscribeProperties EXTENDS FB_IotMqtt5UserProperties [▶ 108]
VAR_OUTPUT
    bError          : BOOL;
    hrErrorCode     : HRESULT;
END_VAR
```

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.

Properties

Name	Type	Access	Description
pUnsubscribeProperties	POINTER TO MqttUnsubscribeProperties	Get	Pointer to an object of type MqttUnsubscribeProperties. When calling FB_IotMqtt5Client.Unsubscribe() it is possible to pass this directly.

Methods

Name	Description
SetUnsubscribeProperties	Allows to set the Unsubscribe properties.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4026.0	IPC or CX (x86, x64, ARM)	Tc3_IotBase (>= v3.4.2.0)

5.1.2.9.6 FB_IotMqtt5UserProperties

The function block enables the definition and handling of UserProperties. UserProperties are key/value pairs that describe optional metadata and can be used, for example, with PublishProperties.

Number of UserProperties

The processing of UserProperties in the PLC real-time task cycle requires a certain amount of time, which is why it is better to keep it low.
The maximum amount of possible UserProperties can be set via the parameter `cMaxMqtt5UserProps` in the parameter list [▶ 111] of the Tc3_IotBase library.

Syntax

Definition:

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

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.

Properties

Name	Type	Access	Description
nUserPropertyCnt	UINT	Get	Specifies the number of UserProperties included.

Methods

Name	Description
AddUserProperty	Adds a UserProperty to the list. The property is specified by its name and value.
ClearUserProperties	Deletes all UserProperties contained in the list.
GetUserPropertyByIdx	Returns the name and value of a UserProperty specified by its position (index) in the list.
GetUserPropertyValueByName	Returns the value of a UserProperty specified by its name.
SetUserProperties	Deletes all UserProperties contained in the list and adds a set of UserProperties specified by an array of objects of type MqttUserProperty.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4026.0	IPC or CX (x86, x64, ARM)	Tc3_IotBase (>= v3.4.2.0)

5.1.3 ETcIotMqttClientState

Syntax

```
TYPE ETcIotMqttClientState :
(
MQTT_ERR_SUCCESS            :=0,
MQTT_ERR_NOMEM              :=1,
MQTT_ERR_PROTOCOL           :=2,
MQTT_ERR_INVAL              :=3,
MQTT_ERR_NO_CONN            :=4,
MQTT_ERR_CONN_REFUSED       :=5,
MQTT_ERR_NOT_FOUND          :=6,
MQTT_ERR_CONN_LOST          :=7,
MQTT_ERR_TLS                :=8,
MQTT_ERR_PAYLOAD_SIZE       :=9,
MQTT_ERR_NOT_SUPPORTED      :=10,
MQTT_ERR_AUTH               :=11,
MQTT_ERR_ACL_DENIED         :=12,
MQTT_ERR_UNKNOWN            :=13,
MQTT_ERR_ERRNO              :=14,
MQTT_ERR_EAI                :=15,
MQTT_ERR_PROXY              :=16,
MQTT_ERR_TLS_CA_NOTFOUND    :=17,
MQTT_ERR_TLS_CERT_NOTFOUND  :=18,
MQTT_ERR_TLS_KEY_NOTFOUND   :=19,
MQTT_ERR_TLS_CA_INVALID     :=20,
MQTT_ERR_TLS_CERT_INVALID   :=21,
MQTT_ERR_TLS_KEY_INVALID    :=22,
MQTT_ERR_TLS_VERIFY_FAIL    :=23,
MQTT_ERR_TLS_SETUP          :=24,
```

```
MQTT_ERR_TLS_HANDSHAKE_FAIL :=25,
MQTT_ERR_TLS_CIPHER_INVALID :=26,
MQTT_ERR_TLS_VERSION_INVALID:=27,
MQTT_ERR_TLS_PSK_INVALID    :=28,
MQTT_ERR_TLS_CRL_NOTFOUND   :=29,
MQTT_ERR_TLS_CRL_INVALID    :=30,
MQTT_ERR_FINALIZE_DISCONNECT:=31,
MQTT_ERR_BIND               :=32,
MQTT_ERR_BIND_ADDR_INUSE    :=33,
MQTT_ERR_BIND_ADDR_INVAL    :=34,
MQTT_ERR_CREATE             :=35,
MQTT_ERR_CREATE_TYPE        :=36,
MQTT_ERR_CONN               :=37,
MQTT_ERR_CONN_TIMEDOUT      :=38,
MQTT_ERR_CONN_HOSTUNREACH   :=39,
MQTT_ERR_TLS_CERT_EXPIRED   :=40,
MQTT_ERR_TLS_CN_MISMATCH    :=41
) DINT;
END_TYPE
```

Parameter

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

5.1.4 Parameter list

Parameters that influence the MQTT Message Queue (`FB_IotMqttMessageQueue`, `FB_IotMqtt5MessageQueue`) as well as the dynamic memory used for the MQTT messages (TwinCAT Router memory):

Name	Type	Default value	Description
cMaxSizeOfMqttMessage	UDINT	102400	Maximum size in bytes of an MQTT message that can be collected in the message queue. If the `FB_IotMqttClient` receives an MQTT message that is too large, it will be completely discarded when trying to store it in the `FB_IotMqttMessageQueue`. If the `FB_IotMqtt5Client` receives such an MQTT message, it will first be reduced by all User Properties when it attempts to store it in the `FB_IotMqtt5MessageQueue`. If this is not sufficient, the MQTT message is discarded completely.
cMaxSizeOfMqttMessageQueue	UDINT	1024000	Maximum size in bytes of an entire message queue including all collected MQTT messages. If the `FB_IotMqttClient` receives a new MQTT message that would exceed the maximum size of the message queue, this message is completely discarded when it is attempted to be stored in the `FB_IotMqttMessageQueue`. If the `FB_IotMqtt5Client` receives such an MQTT message, it waits until there is space in the queue to receive this new message.
cMaxEntriesInMqttMessageQueue	UDINT	1000	Maximum number of messages in a message queue.

Parameters that influence the inputs of an MQTT client (`FB_IotMqttClient, FB_IotMqtt5Client`):

Name	Type	Default value	Description
cSizeOfMqttClientClientId	UDINT	256	Definition of the STRING size in bytes, which specifies the maximum length of the ClientId.
cSizeOfMqttClientHostName	UDINT	256	Definition of the STRING size in bytes, which specifies the maximum length of the host name.
cSizeOfMqttClientTopicPrefix	UDINT	256	Definition of the STRING size in bytes, which specifies the maximum length of the topic prefix.
cSizeOfMqttClientUserName	UDINT	256	Definition of the STRING size in bytes, which specifies the maximum length of the user name.
cSizeOfMqttClientUserPwd	UDINT	256	Definition of the STRING size in bytes, which specifies the maximum length of the user password.
cSizeOfMqttWillTopic	UDINT	256	Definition of the STRING size in bytes, which specifies the maximum length of the Will-Topic.

Parameters that influence the MQTT 5 properties:

Name	Type	Default value	Description
cSizeOfMqtt5ContentType	UDINT	256	Definition of the STRING size in bytes, which specifies the maximum length of the content type.
cSizeOfMqtt5AuthMethod	UDINT	256	Definition of the STRING size in bytes, which specifies the maximum length of the authentication method.
cSizeOfMqtt5AuthData	UDINT	4096	Maximum size in bytes of the authentication data.
cSizeOfMqtt5ServerReference	UDINT	256	Definition of the STRING size in bytes, which specifies the maximum length of the server reference.
cSizeOfMqtt5ReasonString	UDINT	256	Definition of the STRING size in bytes, which specifies the maximum length of the reason string.
cMaxMqtt5UserProps	UINT	20	Maximum number of UserProperties that can be received with an MQTT 5 message or sent with the help of `FB_IotMqtt5UserProperties`. **The processing of UserProperties in the PLC real-time task cycle requires a certain amount of time depending on the quantity, which is why it is better to keep it low.**

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 [▶ 183] 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 [▶ 183] is recommended for JSON documents to which many changes are made (e.g. the cyclic changing of a specific value in the JSON document).

⚠ 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 NewDocument [▶ 159]() or ParseDocument [▶ 159]() methods.

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.

Syntax

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

Outputs

Name	Type
initStatus	HRESULT

Methods

Name	Description
AddArrayMember [▶ 119]	Adds an Array member to a JSON object.
AddBase64Member [▶ 120]	Adds a Base64 member to a JSON object.
AddBoolMember [▶ 121]	Adds a Bool member to a JSON object.
AddDateTimeMember [▶ 122]	Adds a DateTime member to a JSON object.
AddDcTimeMember [▶ 122]	Adds a DcTime member to a JSON object.
AddDoubleMember [▶ 123]	Adds a Double member to a JSON object.
AddFileTimeMember [▶ 124]	Adds a FileTime member to a JSON object.
AddHexBinaryMember [▶ 124]	Adds a HexBinary member to a JSON object.
AddInt64Member [▶ 125]	Adds an Int64 member to a JSON object.
AddIntMember [▶ 126]	Adds an Int member to a JSON object.
AddJsonMember [▶ 126]	Adds a JSON member to a JSON object.
AddNullMember [▶ 127]	Adds a NULL member to a JSON object.
AddObjectMember [▶ 128]	Adds an Object member to a JSON object.
AddStringMember [▶ 128]	Adds a String member to a JSON object.
AddUint64Member [▶ 129]	Adds an UInt64 member to a JSON object.
AddUintMember [▶ 130]	Adds an UInt member to a JSON object.
ArrayBegin [▶ 130]	Returns the first element of an array.
ArrayEnd [▶ 131]	Returns the last element of an array.
ClearArray [▶ 131]	Deletes the contents of an array.
CopyDocument [▶ 132]	Copies the contents of the DOM memory into a variable of data type STRING.
CopyJson [▶ 133]	Extracts a JSON object from a key and stores it in a variable of data type STRING.
CopyString [▶ 134]	Copies the value of a key into a variable of data type STRING.
FindMember [▶ 135]	Searches for a specific property in a JSON document.
FindMemberPath [▶ 136]	Searches for a specific property in a JSON document (path-specific).
GetArraySize [▶ 138]	Returns the number of elements in a JSON array.
GetArrayValue [▶ 138]	Returns the value at the current iterator position of an array.
GetArrayValueByIdx [▶ 139]	Returns the value of an array at a specified index.
GetBase64 [▶ 139]	Decodes a Base64 value from a JSON property.
GetBool [▶ 140]	Returns the value of a property of the data type BOOL.
GetDateTime [▶ 140]	Returns the value of a property of the data type DATE_AND_TIME.
GetDcTime [▶ 140]	Returns the value of a property of the data type DCTIME.
GetDocument [▶ 141]	Returns the content of the DOM memory as the data type STRING(255).
GetDocumentLength [▶ 141]	Returns the length of a JSON document in the DOM memory.
GetDocumentRoot [▶ 142]	Returns the root node of a JSON document in the DOM memory.
GetDouble [▶ 142]	Returns the value of a property of the data type LREAL.
GetFileTime [▶ 142]	Returns the value of a property of the data type DCTIME.
GetHexBinary [▶ 143]	Decodes the HexBinary content of a property and writes it to a certain memory address,
GetInt [▶ 144]	Returns the value of a property of the data type DINT.
GetInt64 [▶ 144]	Returns the value of a property of the data type LINT.
GetJson [▶ 144]	Returns the value of a property of the data type STRING(255).
GetJsonLength [▶ 145]	Returns the length of a property if this is a JSON document.

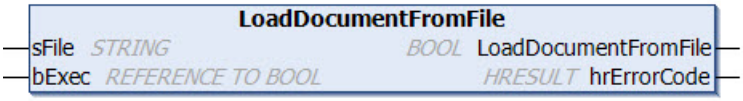
Name	Description
GetMaxDecimalPlaces [▶ 146]	Returns the current setting for MaxDecimalPlaces.
GetMemberName [▶ 146]	Returns the name of a JSON property member at the position of the current iterator,
GetMemberValue [▶ 146]	Returns the value of a JSON property member at the position of the current iterator,
GetString [▶ 147]	Returns the value of a property of the data type STRING(255).
GetStringLength [▶ 148]	Returns the length of a property if its value is a string.
GetType [▶ 148]	Returns the type of a property value.
GetUint [▶ 149]	Returns the value of a property of the data type UDINT.
GetUint64 [▶ 149]	Returns the value of a property of the data type ULINT.
HasMember [▶ 150]	Checks whether a certain property is present in the DOM memory.
IsArray [▶ 150]	Checks whether a given property is an array.
IsBase64 [▶ 151]	Checks whether the value of a given property is of the data type Base64.
IsBool [▶ 151]	Checks whether the value of a given property is of the data type BOOL.
IsDouble [▶ 152]	Checks whether the value of a given property is of the data type Double (PLC: LREAL).
IsFalse [▶ 152]	Checks whether the value of a given property is FALSE.
IsHexBinary [▶ 153]	Checks whether the value of a property is in the HexBinary format.
IsInt [▶ 153]	Checks whether the value of a given property is of the data type Integer (PLC: DINT) .
IsInt64 [▶ 154]	Checks whether the value of a given property is of the data type LINT.
IsISO8601TimeFormat [▶ 154]	Checks whether the value of a given property has a time format according to ISO8601.
IsNull [▶ 154]	Checks whether the value of a given property is NULL.
IsNumber [▶ 155]	Checks whether the value of a given property is a numerical value.
IsObject [▶ 155]	Checks whether the given property is a further JSON object.
IsString [▶ 156]	Checks whether the value of a given property is of the data type STRING.
IsTrue [▶ 156]	Checks whether the value of a given property is TRUE.
IsUint [▶ 157]	Checks whether the value of a given property is of the data type UDINT.
IsUint64 [▶ 157]	Checks whether the value of a given property is of the data type ULINT.
LoadDocumentFromFile [▶ 118]	Loads a JSON document from a file.
MemberBegin [▶ 158]	Returns the first child element below a JSON property.
MemberEnd [▶ 158]	Returns the last child element below a JSON property.
NewDocument [▶ 159]	Generates a new empty JSON document in the DOM memory.
NextArray [▶ 159]	Returns the next element in an array.
NextMember	Returns the next property in a JSON document.
ParseDocument [▶ 159]	Loads a JSON object into the DOM memory for further processing.
PopbackValue	Deletes the element at the end of an array.
PushbackBase64Value [▶ 160]	Appends a Base64 value to the end of an array.
PushbackBoolValue [▶ 161]	Appends a Base64 value to the end of an array.
PushbackDateTimeValue [▶ 162]	Appends a value of the data type DATE_AND_TIME to the end of an array.
PushbackDcTimeValue [▶ 162]	Appends a value of the data type DCTIME to the end of an array.

Name	Description
PushbackDoubleValue [▶ 163]	Appends a value of the data type Double to the end of an array.
PushbackFileTimeValue [▶ 163]	Appends a value of the data type FILETIME to the end of an array.
PushbackHexBinaryValue [▶ 164]	This method appends a HexBinary-coded value to the end of an array.
PushbackInt64Value [▶ 164]	Appends a value of the data type Int64 to the end of an array.
PushbackIntValue [▶ 165]	Appends a value of the data type INT to the end of an array.
PushbackJsonValue [▶ 166]	Appends a JSON document to the end of an array.
PushbackNullValue [▶ 166]	Appends a NULL value to the end of an array.
PushbackStringValue [▶ 167]	Appends a value of the data type String to the end of an array.
PushbackUint64Value [▶ 167]	Appends a value of the data type UInt64 to the end of an array.
PushbackUintValue [▶ 168]	Appends a value of the data type UInt to the end of an array.
RemoveAllMembers [▶ 168]	Removes all child elements from a given property.
RemoveArray [▶ 169]	Deletes the value of the current array iterator.
RemoveMember [▶ 170]	Deletes the property at the current iterator.
RemoveMemberByName [▶ 170]	Removes a child element from a given property.
SaveDocumentToFile [▶ 171]	Saves a JSON document in a file.
SetArray [▶ 173]	Sets the value of a property to the type "Array".
SetBase64 [▶ 173]	Sets the value of a property to a Base64-coded value.
SetBool [▶ 174]	Sets the value of a property to a value of the data type BOOL.
SetDateTime [▶ 174]	Sets the value of a property to a value of the data type DATE_AND_TIME.
SetDcTime [▶ 175]	Sets the value of a property to a value of the data type DCTIME.
SetDouble [▶ 175]	Sets the value of a property to a value of the data type Double.
SetFileTime [▶ 176]	Sets the value of a property to a value of the data type FILETIME.
SetHexBinary [▶ 177]	Sets the value of a property to a HexBinary-coded value.
SetInt [▶ 177]	Sets the value of a property to a value of the data type INT
SetInt64 [▶ 178]	Sets the value of a property to a value of the data type Int64.
SetJson [▶ 178]	Inserts a further JSON document into the value of a property.
SetMaxDecimalPlaces [▶ 179]	Sets the current setting for MaxDecimalPlaces.
SetNull [▶ 179]	Sets the value of a property to the value NULL.
SetObject [▶ 180]	Sets the value of a property to the type "Object".
SetString [▶ 180]	Sets the value of a property to a value of the data type STRING.
SetUint [▶ 181]	Sets the value of a property to a value of the data type UInt.
SetUint64 [▶ 182]	Sets the value of a property to a value of the data type UInt64.

Requirements

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

5.2.1.1.1 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 whether the loading of the file was successful (TRUE) or failed (FALSE) for the duration of one call.

Encoding

Please note that the imported files must be formatted in UTF-8. Formats such as UTF-16 are not supported.

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
```

Return value

Name	Type
LoadDocumentFromFile	BOOL

Inputs

Name	Type
bExec	REFERENCE TO BOOL

Inputs/outputs

Name	Type
sFile	STRING

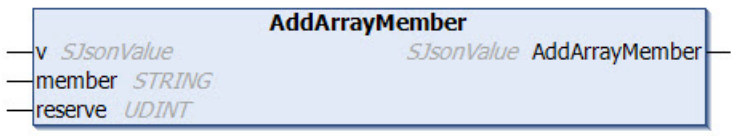
Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

5.2.1.1.2 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
```

Return value

Name	Type
AddArrayMember	SJsonValue

Inputs

Name	Type
v	SJsonValue
reserve	UDINT

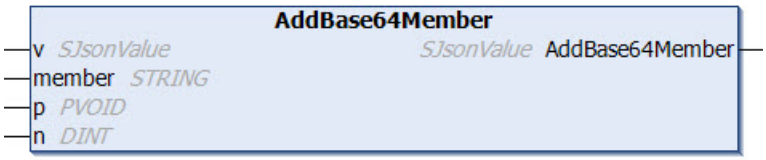
/ Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

5.2.1.1.3 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
```

Return value

Name	Type
AddBase64Member	SJsonValue

Inputs

Name	Type
v	SJsonValue
p	PVOID
n	DINT

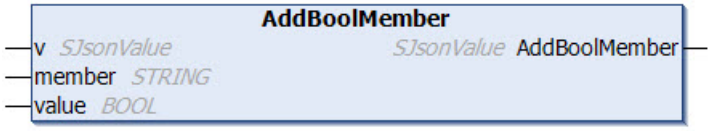
/ Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

5.2.1.1.4 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
```

Return value

Name	Type
AddBoolMember	SJsonValue

Inputs

Name	Type
v	SJsonValue
value	BOOL

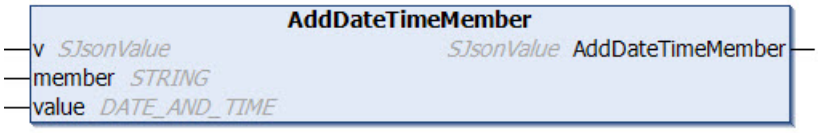
/ Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

5.2.1.1.5 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
```

Return value

Name	Type
AddDateTimeMember	SJsonValue

Inputs

Name	Type
v	SJsonValue
value	DATE_AND_TIME

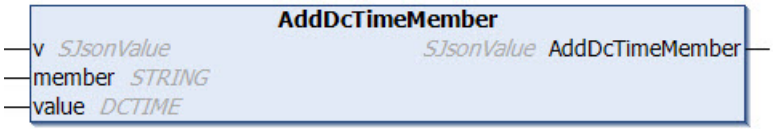
/ Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

5.2.1.1.6 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
```

Return value

Name	Type
AddDcTimeMember	SJsonValue

Inputs

Name	Type
v	SJsonValue
value	DCTIME

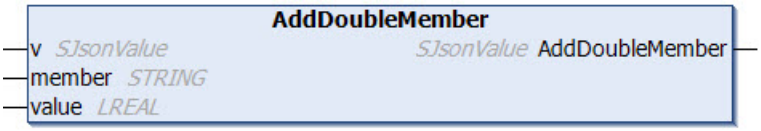
Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

5.2.1.1.7 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
```

Return value

Name	Type
AddDoubleMember	SJsonValue

Inputs

Name	Type
v	SJsonValue
value	LREAL

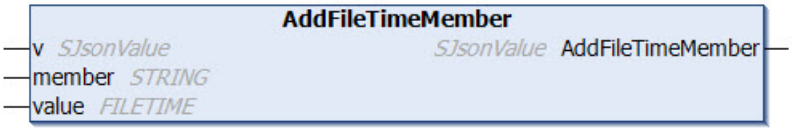
Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

5.2.1.1.8 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
```

Return value

Name	Type
AddFileTimeMember	SJsonValue

Inputs

Name	Type
v	SJsonValue
value	FILETIME

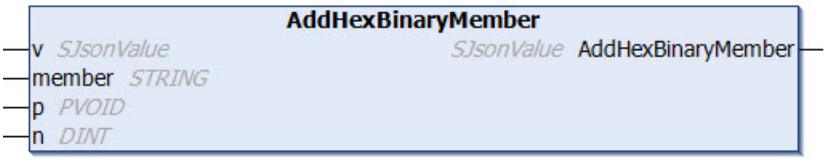
Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

5.2.1.1.9 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
```

Return value

Name	Type
AddHexBinaryMember	SjsonValue

Inputs

Name	Type
v	SjsonValue
p	PVOID
n	DINT

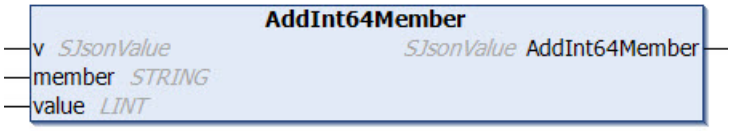
/ Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

5.2.1.1.10 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
```

Return value

Name	Type
AddInt64Member	SJsonValue

Inputs

Name	Type
v	SJsonValue
value	LINT

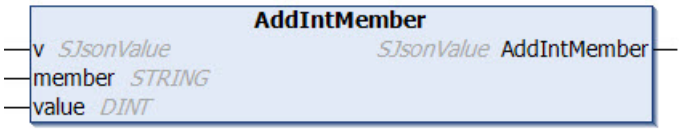
/ Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

5.2.1.1.11 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
```

Return value

Name	Type
AddIntMember	SJsonValue

Inputs

Name	Type
v	SJsonValue
value	DINT

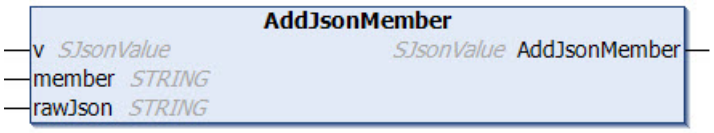
/ Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

5.2.1.1.12 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
```

Return value

Name	Type
AddJsonMember	SJsonValue

Inputs

Name	Type
v	SJsonValue

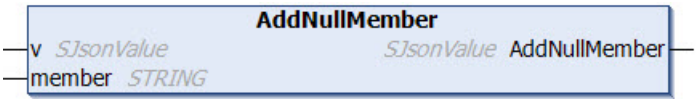
/ Inputs/Outputs

Name	Type
member	STRING
rawJson	STRING

Sample call:

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

5.2.1.1.13 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
```

Return value

Name	Type
AddNullMember	SJsonValue

Inputs

Name	Type
v	SJsonValue

/ Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

5.2.1.1.14 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
```

Return value

Name	Type
AddObjectMember	SJsonValue

Inputs

Name	Type
v	SJsonValue

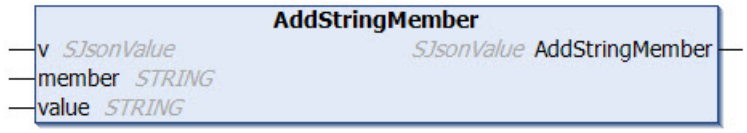
/ Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

5.2.1.1.15 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
```

Return value

Name	Type
AddStringMember	SJsonValue

Inputs

Name	Type
v	SJsonValue

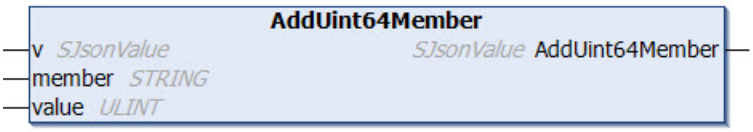
/ Inputs/Outputs

Name	Type
member	STRING
value	STRING

Sample call:

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

5.2.1.1.16 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
```

Return value

Name	Type
AddUint64Member	SJsonValue

Inputs

Name	Type
v	SJsonValue
value	ULINT

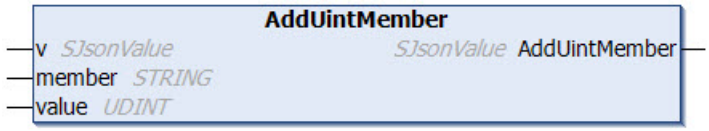
/ Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

5.2.1.1.17 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
```

Return value

Name	Type
AddUintMember	SJsonValue

Inputs

Name	Type
v	SJsonValue
value	UDINT

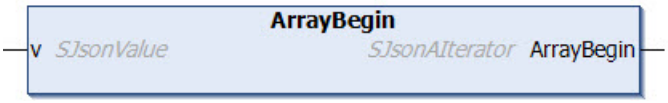
/ Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

5.2.1.1.18 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
```

Return value

Name	Type
ArrayBegin	SJsonAIterator

Inputs

Name	Type
v	SJsonValue

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 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
```

Return value

Name	Type
ArrayEnd	SJsonAIterator

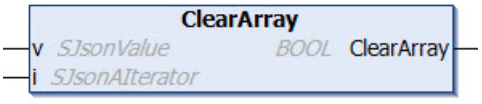
Inputs

Name	Type
v	SJsonValue

Sample call:

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

5.2.1.1.20 ClearArray



This method deletes the content of an array.

Syntax

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

Return value

Name	Type
ClearArray	BOOL

Inputs

Name	Type
v	SJsonValue
i	SJsonAIterator

Sample call:

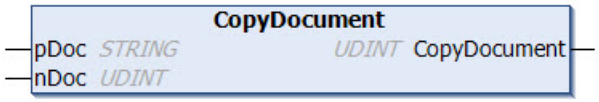
The following JSON document is to be loaded into the DOM memory:

```
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
```

5.2.1.1.21 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 destination 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
```

Return value

Name	Type
CopyDocument	UDINT

Inputs

Name	Type
nDoc	DINT

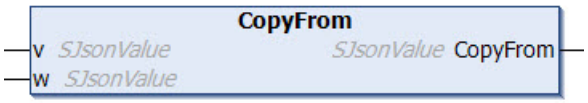
/ Inputs/Outputs

Name	Type
pDoc	STRING

Sample call:

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

5.2.1.1.22 CopyFrom



Syntax

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

Return value

Name	Type
CopyFrom	SJsonValue

Inputs

Name	Type
v	SJsonValue
w	SJsonValue

5.2.1.1.23 CopyJson

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

Syntax

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

Return value

Name	Type
CopyJson	UDINT

Inputs

Name	Type
v	SJsonValue

Inputs/Outputs

Name	Type
pDoc	STRING
nDoc	STRING

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"}
```

5.2.1.1.24 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 destination 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
```

Return value

Name	Type
CopyString	UDINT

Inputs

Name	Type
v	SJsonValue

/ Inputs/Outputs

Name	Type
pStr	STRING
nStr	UDINT

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
```

5.2.1.1.25 ExceptionRaised

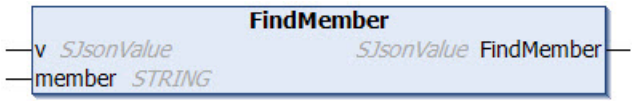
Syntax

```
METHOD ExceptionRaised : BOOL
```

Return value

Name	Type
ExceptionRaised	BOOL

5.2.1.1.26 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
```

Return value

Name	Type
FindMember	SJsonValue

Inputs

Name	Type
v	SJsonValue

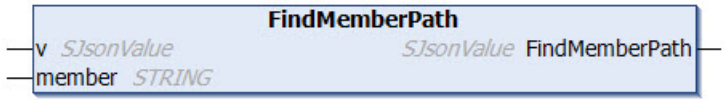
/ Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

5.2.1.1.27 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
```

Return value

Name	Type
FindMemberPath	SJsonValue

Inputs

Name	Type
v	SJsonValue

/ Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

Accessing nested objects works according to the scheme *a/b/c*, to find a variable in a JSON hierarchy. The call for the variable *c* of the following JSON document is:

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

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

Support for arrays

The method supports JSON documents with arrays from TwinCAT version >3.1.4024.35. The # character can be used to access elements of an array.

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

```
[
  {
  "First": 4
  },
  {
  "Second": 12,
  "Third": [
    1,
    2,
    3
    ],
   "Fourth": {
    "a": true
    }
  },
]
```

The example call accesses the second element of the outer array (#1), then the third element of the array under the sub-element *Third*.

Treatment of special cases

Inserting the ~ character provides special treatment within a path. The following table lists the different possibilities.

Expression	Result	Sample
~0	~ is used as a character in the string.	Test/Hello~0123 becomes Test/Hello~123
~1	The expression is replaced by the character **/**, which is not interpreted as a separator, but as part of the string.	Test/Hello~1123 becomes Test/Hello/123
~2	The expression is replaced by the character **#** , which is not interpreted as an array index, but as part of the string.	Test/Hello~2123 becomes Test/Hello**#**123

5.2.1.1.28 GetArraySize

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

Syntax

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

Return value

Name	Type
GetArraySize	UDINT

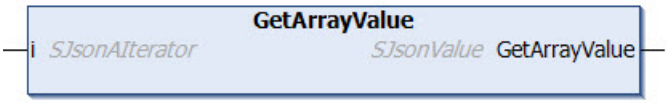
Inputs

Name	Type
v	SJsonValue

Sample call:

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

5.2.1.1.29 GetArrayValue



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

Syntax

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

Return value

Name	Type
GetArrayValue	SJsonValue

Inputs

Name	Type
i	SJsonAIterator

Sample call:

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

5.2.1.1.30 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
```

Return value

Name	Type
GetArrayValueByIdx	SJsonValue

Inputs

Name	Type
v	SJsonValue
idx	UDINT

Sample call:

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

5.2.1.1.31 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
```

Return value

Name	Type
GetBase64	DINT

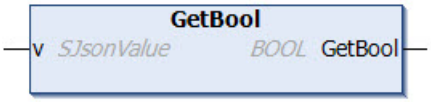
Inputs

Name	Type
v	SJsonValue
p	PVOID
n	DINT

Sample call:

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

5.2.1.1.32 GetBool



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

Syntax

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

Return value

Name	Type
GetBool	BOOL

Inputs

Name	Type
v	SJsonValue

5.2.1.1.33 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
```

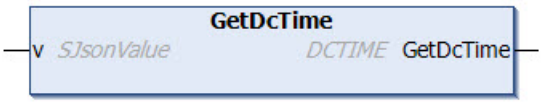
Return value

Name	Type
GetDateTime	DATE_AND_TIME

Inputs

Name	Type
v	SJsonValue

5.2.1.1.34 GetDcTime



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

Syntax

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

Return value

Name	Type
GetDcTime	DCTIME

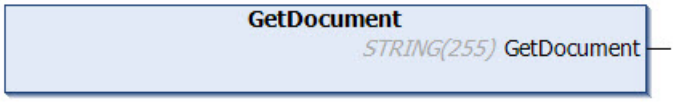
Inputs

Name	Type
v	SJsonValue

Sample call:

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

5.2.1.1.35 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 [▶ 132]() must be used.

Syntax

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

Return value

Name	Type
GetDocument	STRING(255)

Sample call:

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

5.2.1.1.36 GetDocumentLength

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

Syntax

```
METHOD GetDocumentLength: UDINT
```

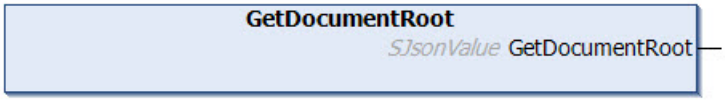
Return value

Name	Type
GetDocumentLength	UDINT

Sample call:

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

5.2.1.1.37 GetDocumentRoot



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

Syntax

```
METHOD GetDocumentRoot : SJsonValue
```

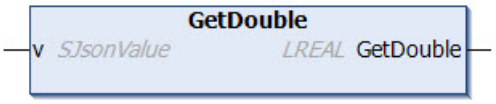
Return value

Name	Type
GetDocumentRoot	SJsonValue

Sample call:

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

5.2.1.1.38 GetDouble



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

Syntax

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

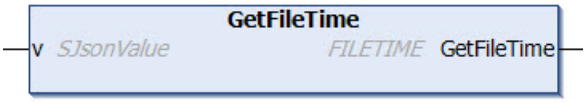
Return value

Name	Type
GetDouble	LREAL

Inputs

Name	Type
v	SJsonValue

5.2.1.1.39 GetFileTime



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

Syntax

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

Return value

Name	Type
GetFileTime	FILETIME

Inputs

Name	Type
v	SJsonValue

Sample call:

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

5.2.1.1.40 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
```

Return value

Name	Type
GetHexBinary	DINT

Inputs

Name	Type
v	SJsonValue
p	PVOID
n	DINT

Sample call:

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

5.2.1.1.41 GetInt

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

Syntax

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

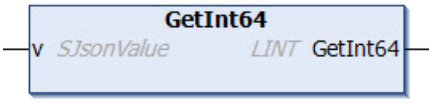
Return value

Name	Type
GetInt	DINT

Inputs

Name	Type
v	SJsonValue

5.2.1.1.42 GetInt64



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

Syntax

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

Return value

Name	Type
GetInt64	LINT

Inputs

Name	Type
v	SJsonValue

5.2.1.1.43 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 [▶ 133]() must be used.

Syntax

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

Return value

Name	Type
GetJson	STRING(255)

Inputs

Name	Type
v	SJsonValue

Sample call:

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

5.2.1.1.44 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
```

Return value

Name	Type
GetJsonLength	UDINT

Inputs

Name	Type
v	SJsonValue

Sample call:

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

5.2.1.1.45 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
```

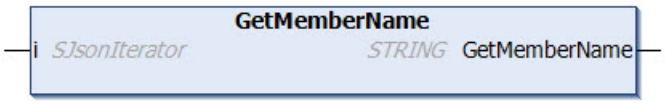
Return value

Name	Type
GetMaxDecimalPlaces	DINT

Sample call:

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

5.2.1.1.46 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(255)
VAR_INPUT
  i : SJsonIterator;
END_VAR
```

Return value

Name	Type
GetMemberName	STRING(255)

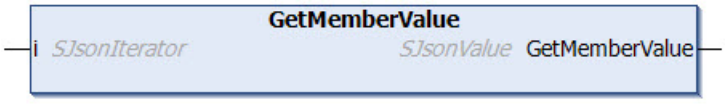
Inputs

Name	Type
i	SJsonIterator

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
```

5.2.1.1.47 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
```

Return value

Name	Type
GetMemberValue	SJsonValue

Inputs

Name	Type
i	SJsonIterator

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
```

5.2.1.1.48 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 [▶ 134]() must be used.

Syntax

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

Return value

Name	Type
GetString	STRING(255)

Inputs

Name	Type
v	SJsonValue

5.2.1.1.49 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
```

Return value

Name	Type
GetStringLength	UDINT

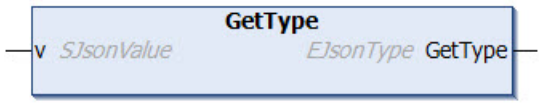
Inputs

Name	Type
v	SJsonValue

Sample call:

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

5.2.1.1.50 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 GetType : EJsonType
VAR_INPUT
  v            : SJsonValue
END_VAR

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

Return value

Name	Type
GetType	EJsonType

Inputs

Name	Type
v	SJsonValue

Sample call:

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

5.2.1.1.51 GetUint

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

Syntax

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

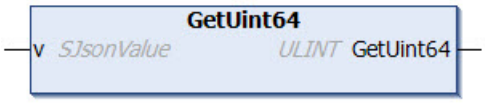
Return value

Name	Type
GetUint	UDINT

Inputs

Name	Type
v	SJsonValue

5.2.1.1.52 GetUint64



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

Syntax

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

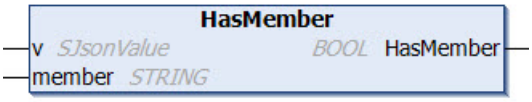
Return value

Name	Type
GetUint64	ULINT

Inputs

Name	Type
v	SJsonValue

5.2.1.1.53 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
```

Return value

Name	Type
HasMember	BOOL

Inputs

Name	Type
v	SJsonValue

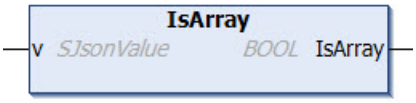
/ Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

5.2.1.1.54 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
```

Return value

Name	Type
IsArray	BOOL

Inputs

Name	Type
v	SJsonValue

5.2.1.1.55 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
```

Return value

Name	Type
IsBase64	BOOL

Inputs

Name	Type
v	SJsonValue

5.2.1.1.56 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
```

Return value

Name	Type
IsBool	BOOL

Inputs

Name	Type
v	SJsonValue

5.2.1.1.57 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
```

Return value

Name	Type
IsDouble	BOOL

Inputs

Name	Type
v	SJsonValue

5.2.1.1.58 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
```

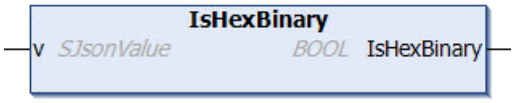
Return value

Name	Type
IsFalse	BOOL

Inputs

Name	Type
v	SJsonValue

5.2.1.1.59 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
```

Return value

Name	Type
IsHexBinary	BOOL

Inputs

Name	Type
v	SJsonValue

Sample call:

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

5.2.1.1.60 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
```

Return value

Name	Type
IsInt	BOOL

Inputs

Name	Type
v	SJsonValue

5.2.1.1.61 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
```

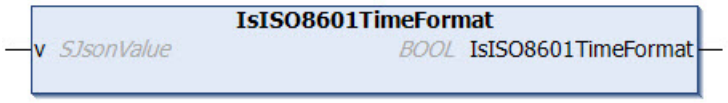
Return value

Name	Type
IsInt64	BOOL

Inputs

Name	Type
v	SJsonValue

5.2.1.1.62 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
```

Return value

Name	Type
IsISO8601TimeFormat	BOOL

Inputs

Name	Type
v	SJsonValue

5.2.1.1.63 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
```

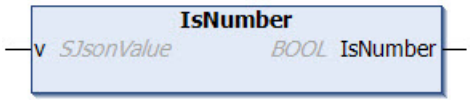
Return value

Name	Type
IsNull	BOOL

Inputs

Name	Type
v	SJsonValue

5.2.1.1.64 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
```

Return value

Name	Type
IsNumber	BOOL

Inputs

Name	Type
v	SJsonValue

5.2.1.1.65 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
```

Return value

Name	Type
IsObject	BOOL

Inputs

Name	Type
v	SJsonValue

5.2.1.1.66 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
```

Return value

Name	Type
IsString	BOOL

Inputs

Name	Type
v	SJsonValue

5.2.1.1.67 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
```

Return value

Name	Type
IsTrue	BOOL

Inputs

Name	Type
v	SJsonValue

5.2.1.1.68 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
```

Return value

Name	Type
IsUint	BOOL

Inputs

Name	Type
v	SJsonValue

5.2.1.1.69 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
```

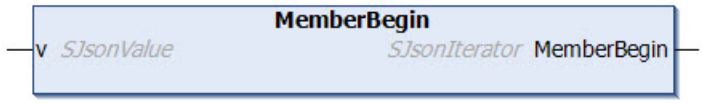
Return value

Name	Type
IsUint64	BOOL

Inputs

Name	Type
v	SJsonValue

5.2.1.1.70 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
```

Return value

Name	Type
MemberBegin	SJsonIterator

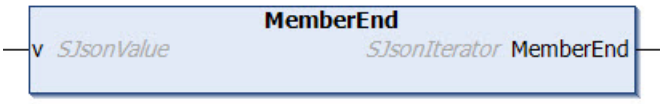
Inputs

Name	Type
v	SJsonValue

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
```

5.2.1.1.71 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
```

Return value

Name	Type
MemberEnd	SJsonIterator

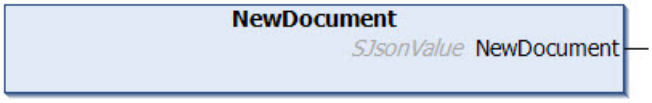
Inputs

Name	Type
v	SJsonValue

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
```

5.2.1.1.72 NewDocument



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

Syntax

```
METHOD NewDocument : SJsonValue
```

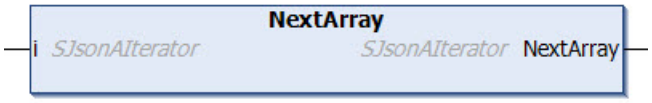
Sample call:

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

Return value

Name	Type
NewDocument	SJsonValue

5.2.1.1.73 NextArray



Syntax

```
METHOD NextArray : SJsonAIterator
VAR_INPUT
  v : SJsonAIterator;
END_VAR
```

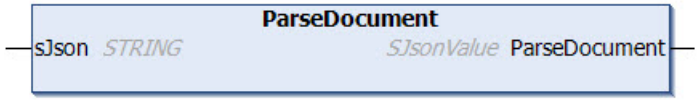
Return value

Name	Type
NextArray	SJsonAIterator

Inputs

Name	Type
i	SJsonAIterator

5.2.1.1.74 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
```

Return value

Name	Type
ParseDocument	SJsonValue

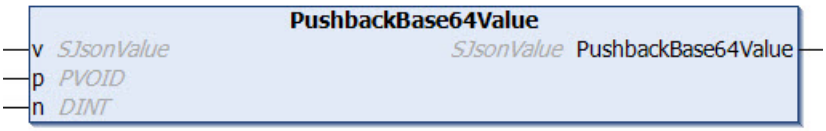
Inputs

Name	Type
sJson	STRING

Sample call:

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

5.2.1.1.75 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
```

Return value

Name	Type
PushbackBase64Value	SJsonValue

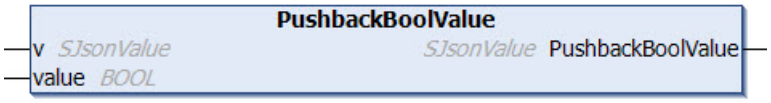
Inputs

Name	Type
v	sJsonValue
p	PVOID
n	DINT

Sample call:

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

5.2.1.1.76 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
```

Return value

Name	Type
PushbackBoolValue	SJsonValue

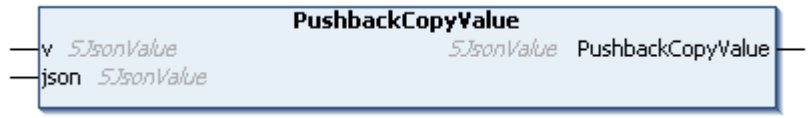
Inputs

Name	Type
v	sJsonValue
value	BOOL

Sample call:

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

5.2.1.1.77 PushbackCopyValue



This method adds a JSON document from the memory of a FB_JsonDomParser [▶ 113] to the end of an array.

Syntax

```
METHOD PushbackCopyValue : SJsonValue
VAR_INPUT
  v       : SJsonValue;
  json    : SJsonValue;
END_VAR
```

Return value

Name	Type
PushbackCopyValue	SJsonValue

Inputs

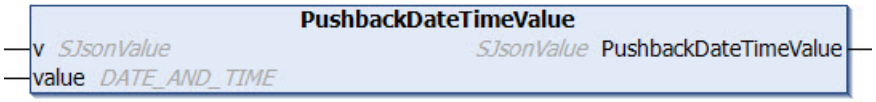
Name	Type
v	SJsonValue
json	SJsonValue

Sample call:

```
jsonCopy:=fbJsonCreate.NewDocument();
fbJsonCreate.AddIntMember(jsonCopy,'a',1);
fbJsonCreate.AddIntMember(jsonCopy,'b',2);

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

5.2.1.1.78 PushbackDateTimeValue



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
```

Return value

Name	Type
PushbackDateTimeValue	SJsonValue

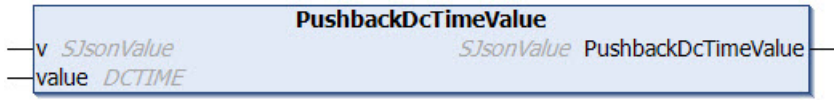
Inputs

Name	Type
v	sJsonValue
value	DATE_AND_TIME

Sample call:

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

5.2.1.1.79 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
```

Return value

Name	Type
PushbackDcTimeValue	SJsonValue

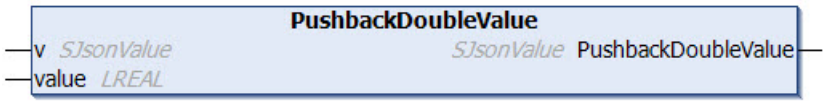
Inputs

Name	Type
v	sJsonValue
value	DCTIME

Sample call:

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

5.2.1.1.80 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
```

Return value

Name	Type
PushbackDoubleValue	SJsonValue

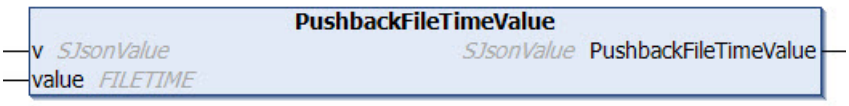
Inputs

Name	Type
v	sJsonValue
value	LREAL

Sample call:

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

5.2.1.1.81 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
```

Return value

Name	Type
PushbackFileTimeValue	SJsonValue

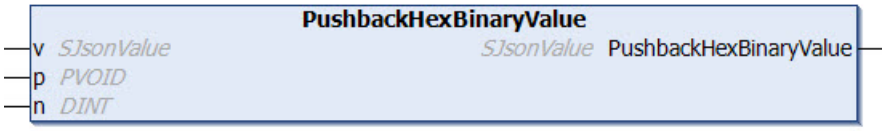
Inputs

Name	Type
v	sJsonValue
value	FILETIME

Sample call:

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

5.2.1.1.82 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
```

Return value

Name	Type
PushbackHexBinaryValue	SJsonValue

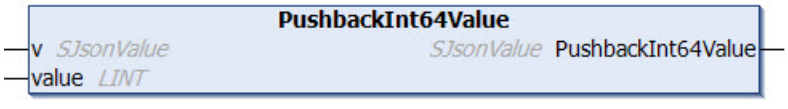
Inputs

Name	Type
v	sJsonValue
p	PVOID
n	DINT

Sample call:

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

5.2.1.1.83 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
```

Return value

Name	Type
PushbackInt64Value	SJsonValue

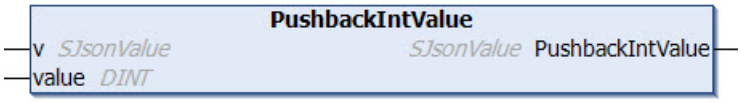
Inputs

Name	Type
v	sJsonValue
value	LINT

Sample call:

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

5.2.1.1.84 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
```

Return value

Name	Type
PushbackIntValue	SJsonValue

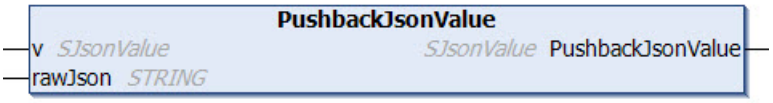
Inputs

Name	Type
v	sJsonValue
value	DINT

Sample call:

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

5.2.1.1.85 PushbackJsonValue



This method appends a JSON document that exists as a STRING in the PLC to the end of an array.

Syntax

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

Return value

Name	Type
PushbackJsonValue	SJsonValue

Inputs

Name	Type
v	SJsonValue

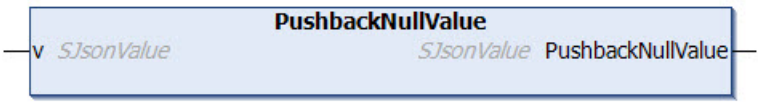
/ Inputs/Outputs

Name	Type
rawJson	STRING

Sample call:

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

5.2.1.1.86 PushbackNullValue



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

Syntax

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

Return value

Name	Type
PushbackNullValue	SJsonValue

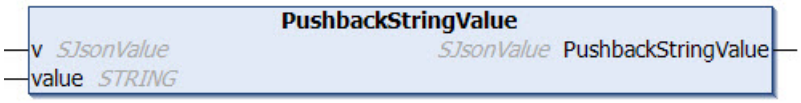
Inputs

Name	Type
v	SJsonValue

Sample call:

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

5.2.1.1.87 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
```

Return value

Name	Type
PushbackStringValue	SJsonValue

Inputs

Name	Type
v	SJsonValue

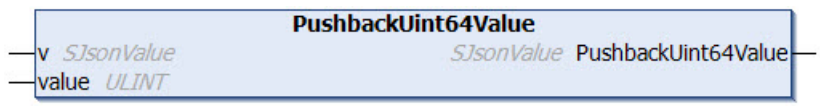
/ Inputs/Outputs

Name	Type
value	STRING

Sample call:

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

5.2.1.1.88 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
```

Return value

Name	Type
PushbackUint64Value	SJsonValue

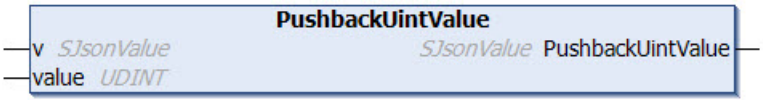
Inputs

Name	Type
v	SJsonValue
value	ULINT

Sample call:

```
jsonDoc   := fbJson.ParseDocument(sExistingJsonDocument);
jsonArray := fbJson.FindMember(jsonDoc, 'array');
jsonValue := fbJson.PushbackUint64Value(jsonArray, 42);
```

5.2.1.1.89 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
```

Return value

Name	Type
PushbackUintValue	SJsonValue

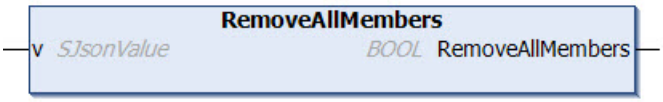
Inputs

Name	Type
v	SJsonValue
value	UDINT

Sample call:

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

5.2.1.1.90 RemoveAllMembers



This method removes all child elements from a given property.

Syntax

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

Return value

Name	Type
RemoveAllMembers	BOOL

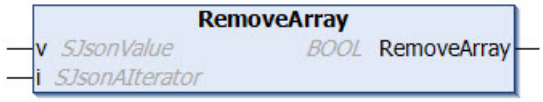
Inputs

Name	Type
v	SJsonValue

Sample call:

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

5.2.1.1.91 RemoveArray



This method deletes the value of the current array iterator.

Syntax

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

Return value

Name	Type
RemoveArray	BOOL

Inputs

Name	Type
v	SJsonValue
i	SJsonAIterator

Sample call:

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

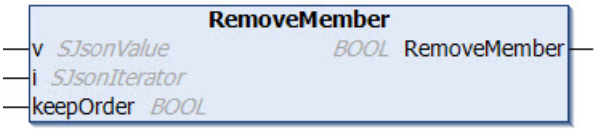
```
sMessage := '{"serialNumber":"123","batteryVoltage":"1547mV","clickType":"SINGLE", "array":
["Hello",2,3]}';
```

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
```

5.2.1.1.92 RemoveMember



This method deletes the property at the current iterator.

Syntax

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

Return value

Name	Type
RemoveMember	BOOL

Inputs

Name	Type
v	SJsonValue
i	SJsonIterator
keepOrder	BOOL

Sample call:

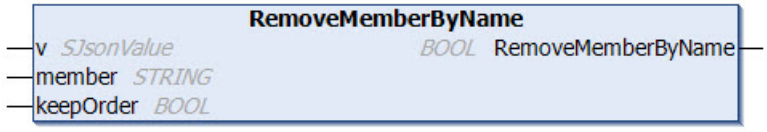
The following JSON document is to be loaded into the DOM memory:

```
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
```

5.2.1.1.93 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
```

Return value

Name	Type
RemoveMemberByName	BOOL

Inputs

Name	Type
v	SJsonValue
keepOrder	BOOL

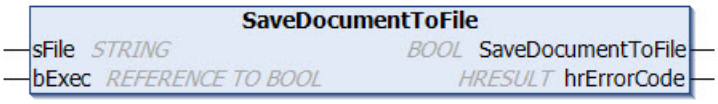
/ Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

5.2.1.1.94 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
```

Return value

Name	Type
SaveDocumentToFile	BOOL

Inputs

Name	Type
bExec	REFERENCE TO BOOL

/ Inputs/Outputs

Name	Type
sFile	STRING

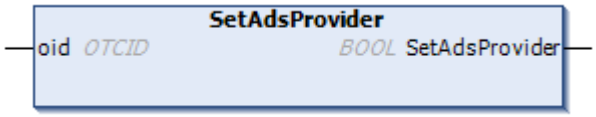
Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

5.2.1.1.95 SetAdsProvider



This method sets the context in which the file accesses are processed when loading and saving the JSON documents. If no ADS provider is specified, the current task is used.

Syntax

```
METHOD SetAdsProvider : BOOL
VAR_INPUT
  oid : OTCID;
END_VAR
```

Return value

Name	Type	Description
SetAdsProvider	BOOL	The method returns TRUE if the setting of the ADS provider was successful.

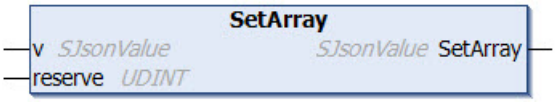
Inputs

Name	Type	Description
oid	OTCID	Object ID of the task.

Sample call:

```
bTest   := fbJson.SetAdsProvider(02010030);
```

5.2.1.1.96 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
```

Return value

Name	Type
SetArray	SJsonValue

Inputs

Name	Type
v	SJsonValue

Sample call:

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

5.2.1.1.97 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
```

Return value

Name	Type
SetBase64	SJsonValue

Inputs

Name	Type
v	SJsonValue
p	PVOID
n	DINT

Sample call:

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

5.2.1.1.98 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
```

Return value

Name	Type
SetBool	SJsonValue

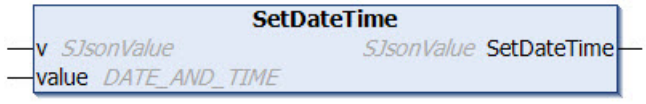
Inputs

Name	Type
v	SJsonValue
value	BOOL

Sample call:

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

5.2.1.1.99 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
```

Return value

Name	Type
SetDateTime	SJsonValue

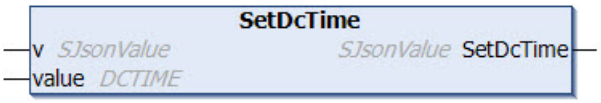
Inputs

Name	Type
v	SJsonValue
value	DATE_AND_TIME

Sample call:

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

5.2.1.1.100 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
```

Return value

Name	Type
SetDcTime	SJsonValue

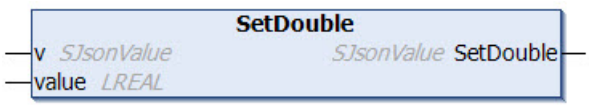
Inputs

Name	Type
v	SJsonValue
value	DCTIME

Sample call:

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

5.2.1.1.101 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
```

Return value

Name	Type
SetDouble	SJsonValue

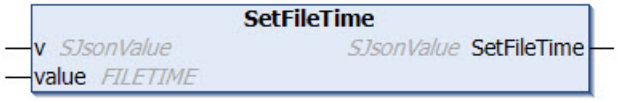
Inputs

Name	Type
v	SJsonValue
value	LREAL

Sample call:

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

5.2.1.1.102 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
```

Return value

Name	Type
SetFileTime	SJsonValue

Inputs

Name	Type
v	SJsonValue
value	FILETIME

Sample call:

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

5.2.1.1.103 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
```

Return value

Name	Type
SetHexBinary	SJsonValue

Inputs

Name	Type
v	SJsonValue
p	PVOID
n	DINT

Sample call:

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

5.2.1.1.104 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
```

Return value

Name	Type
SetInt	SJsonValue

Inputs

Name	Type
v	SJsonValue
value	DINT

Sample call:

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

5.2.1.1.105 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
```

Return value

Name	Type
SetInt64	SJsonValue

Inputs

Name	Type
v	SJsonValue
value	LINT

Sample call:

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

5.2.1.1.106 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
```

Return value

Name	Type
SetJson	SJsonValue

Inputs

Name	Type
v	SJsonValue

/ Inputs/Outputs

Name	Type
rawJson	STRING

Sample call:

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

5.2.1.1.107 SetMaxDecimalPlaces

This function determines the number of decimal places after which a floating-point number is truncated.

Syntax

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

Inputs

Name	Type
dp	DINT

Sample call:

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

5.2.1.1.108 SetNull

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

Syntax

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

Return value

Name	Type
SetNull	SJsonValue

Inputs

Name	Type
v	SJsonValue

Sample call:

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

5.2.1.1.109 SetObject

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

Syntax

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

Return value

Name	Type
SetObject	SJsonValue

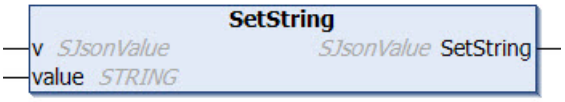
Inputs

Name	Type
v	SJsonValue

Sample call:

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

5.2.1.1.110 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
```

Return value

Name	Type
SetString	SJsonValue

Inputs

Name	Type
v	SJsonValue

/ Inputs/Outputs

Name	Type
value	STRING

Sample call:

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

5.2.1.1.111 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
```

Return value

Name	Type
SetUint	SJsonValue

Inputs

Name	Type
v	SJsonValue
value	UDINT

Sample call:

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

5.2.1.1.112 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
```

Return value

Name	Type
SetUint64	SJsonValue

Inputs

Name	Type
v	SJsonValue
value	ULINT

Sample call:

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

5.2.1.1.113 Swap

This method swaps the content of two pointers.

Syntax

```
METHOD Swap : BOOL
VAR_INPUT
  v     : SJsonValue;
  w     : SJsonValue;
END_VAR;
END_VAR
```

Return value

Name	Type
Swap	BOOL

Inputs

Name	Type
v	SJsonValue
w	SJsonValue

5.2.1.2 FB_JsonDynDomParser

This function block is derived from the same internal function block as FB_JsonDomParser [▶ 113] and thus offers the same interface.

The two derived function blocks differ only in their internal memory management. The FB_JsonDynDomParser is optimized for JSON documents where many changes are made and the JSON document is not released in between. It releases the allocated memory after an action has been executed, e.g. in the methods SetObject() or SetJson(). This flexibility results in greater overhead, allowing the FB_JsonDomParser [▶ 113] to improve the performance.

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.

Syntax

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

Outputs

Name	Type
initStatus	HRESULT

Methods

All methods can be found in FB_JsonDomParser [▶ 113].

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

Syntax

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

Outputs

Name	Type
initStatus	HRESULT

Methods

Name	Description
DecodeBase64 [▶ 184]	Converts a Base64-formated string to binary data.
DecodeDateTime [▶ 185]	Creates a PLC variable of the data type DATE_AND_TIME or DT from a standardized ISO8601 time format (e.g. YYYY-MM-DDThh:mm:ss).
DecodeDcTime [▶ 186]	Creates a PLC variable of the data type DCTIME from a standardized ISO8601 time format (e.g. YYYY-MM-DDThh:mm:ss).
DecodeFileTime [▶ 186]	Creates a PLC variable of the data type FILETIME from a standardized ISO8601 time format (e.g. YYYY-MM-DDThh:mm:ss).
DecodeHexBinary [▶ 187]	Converts a string containing hexadecimal values into binary data.
IsBase64 [▶ 188]	Checks whether the transferred string corresponds to the Base64 format.
IsHexBinary [▶ 188]	Checks whether the transferred string consists of hexadecimal values.
IsISO8601TimeFormat [▶ 189]	Checks whether the transferred string corresponds to the standardized ISO8601 time format.
Parse [▶ 189]	Starts the SAX reader parsing process.
ParseValues [▶ 190]	Starts the SAX reader parsing process.

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-formated 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
```

Return value

Name	Type
DecodeBase64	BOOL

Inputs

Name	Type
sBase64	STRING
pBytes	POINTER TO BYTE
nBytes	REFERENCE TO DINT

Outputs

Name	Type
hrErrorCode	HRESULT

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 01.01.1970. 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
```

Return value

Name	Type
DecodeDateTime	BOOL

Inputs/Outputs

Name	Type
sDT	STRING

Outputs

Name	Type
nDT	DATE_AND_TIME
hrErrorCode	HRESULT

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 01.01.2000. 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
```

Return value

Name	Type
DecodeDcTime	BOOL

Inputs/Outputs

Name	Type
sDC	STRING

Outputs

Name	Type
nDC	DCTIME
hrErrorCode	HRESULT

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 01.01.1601 – 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
```

Return value

Name	Type
DecodeDateTime	BOOL

/ Inputs/Outputs

Name	Type
sFT	STRING

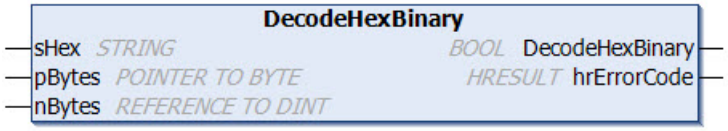
Outputs

Name	Type
nFT	FILETIME
hrErrorCode	HRESULT

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
```

Return value

Name	Type
DecodeHexBinary	BOOL

Inputs

Name	Type
pBytes	POINTER TO BYTE
nBytes	REFERENCE TO DINT

/ Inputs/Outputs

Name	Type
sHex	STRING

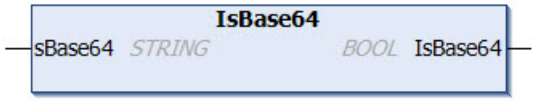
Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

```
bSuccess := fbJson.DecodeHexBinary('ABCEF93A', ADR(byteArray), byteArraySize);
```

5.2.1.3.6 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
```

Return value

Name	Type
IsBase64	BOOL

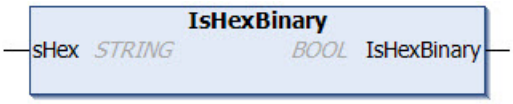
/ Inputs/Outputs

Name	Type
sBase64	STRING

Sample call:

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

5.2.1.3.7 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
```

Return value

Name	Type
IsHexBinary	BOOL

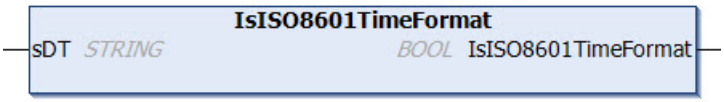
Inputs/Outputs

Name	Type
sHex	STRING

Sample call:

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

5.2.1.3.8 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
```

Return value

Name	Type
IsISO8601TimeFormat	BOOL

Inputs/Outputs

Name	Type
sDT	STRING

Sample call:

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

5.2.1.3.9 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
```

Return value

Name	Type
Parse	BOOL

Inputs

Name	Type
ipHdl	ITcJsonSaxHandler

/ Inputs/Outputs

Name	Type
sJson	STRING

Outputs

Name	Type
hrErrorCode	HRESULT

5.2.1.3.10 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
```

Return value

Name	Type
ParseValues	BOOL

Inputs

Name	Type
ipHdl	ITcJsonSaxValues

/ Inputs/Outputs

Name	Type
sJson	STRING

Outputs

Name	Type
hrErrorCode	HRESULT

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_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.

Syntax

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

Outputs

Name	Type
initStatus	HRESULT

Methods

Name	Description
AddBase64 [▶ 194]	Adds a value of the data type Base64 to a property.
AddBool [▶ 194]	Adds a value of the data type BOOL to a property.
AddDateTime [▶ 195]	Adds a value of the data type DATE_AND_TIME to a property.
AddDcTime [▶ 195]	Adds a value of the data type DCTIME to a property.
AddDint [▶ 195]	Adds a value of the data type DINT to a property.
AddFileTime [▶ 196]	Adds a value of the data type FILETIME to a property.
AddHexBinary [▶ 196]	Adds a HexBinary value to a property.
AddKey [▶ 197]	Adds a new property key at the current position of the SAX Writer.
AddKeyBool [▶ 197]	Creates a new property key and at the same time a value of the data type BOOL.
AddKeyDateTime [▶ 198]	Creates a new property key and at the same time a value of the data type DATE_AND_TIME.
AddKeyDcTime [▶ 198]	Creates a new property key and at the same time a value of the data type DCTIME.
AddKeyFileTime [▶ 199]	Creates a new property key and at the same time a value of the data type FILETIME.
AddKeyLreal [▶ 199]	Creates a new property key and at the same time a value of the data type LREAL.
AddKeyNull [▶ 200]	Creates a new property key and initializes its value with null.
AddKeyNumber [▶ 200]	Creates a new property key and at the same time a value of the data type DINT.
AddKeyString [▶ 201]	Creates a new property key and at the same time a value of the data type STRING.
AddLint [▶ 201]	Adds a value of the data type LINT to a property.
AddLreal [▶ 202]	Adds a value of the data type LREAL to a property.
AddNull [▶ 202]	Adds the value null to a property.
AddRawArray [▶ 202]	Adds a valid JSON array to a given property as a value.
AddRawObject [▶ 203]	Adds a valid JSON object to a given property as a value.
AddReal [▶ 203]	Adds a value of the data type REAL to a property.
AddString [▶ 204]	Adds a value of the data type STRING to a property.
AddUdint [▶ 204]	Adds a value of the data type UDINT to a property.
AddUlint [▶ 204]	Adds a value of the data type ULINT to a property.
CopyDocument [▶ 205]	Copies the contents of the JSON object currently created with the SAX Writer into a target variable of the data type STRING.
EndArray [▶ 206]	Creates the end of a started JSON array and inserts it at the current position of the SAX Writer.
EndObject [▶ 206]	Creates the end of a started JSON object and inserts it at the current position of the SAX Writer.
GetDocument [▶ 206]	Returns the content of the JSON object that is currently created with the SAX Writer and returns it as data type STRING(255).
GetDocumentLength [▶ 207]	Returns the length of the JSON object that is currently created with the SAX Writer and returns it as data type UDINT.
GetMaxDecimalPlaces [▶ 207]	
IsComplete	
ResetDocument [▶ 208]	Resets the JSON object currently created with the SAX Writer.
SetMaxDecimalPlaces [▶ 208]	

Name	Description
StartArray [▶ 208]	Creates the start of a new JSON array and inserts it at the current position of the SAX Writer.
StartObject [▶ 209]	Creates the start of a new JSON object and inserts it at the current position of the SAX Writer.

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() [▶ 197].

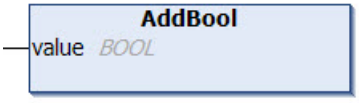
Syntax

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

Inputs

Name	Type
pBytes	POINTER TO BYTE
nBytes	DINT

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() [▶ 197].

Syntax

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

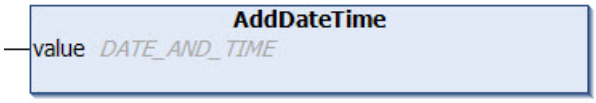
Inputs

Name	Type
value	BOOL

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() [▶ 197].

Syntax

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

Inputs

Name	Type
value	DATE_AND_TIME

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() [▶ 197].

Syntax

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

Inputs

Name	Type
value	DCTIME

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() [▶ 197].

Syntax

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

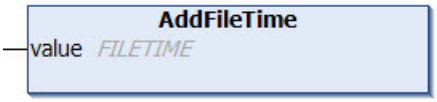
Inputs

Name	Type
value	DINT

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() [▶ 197].

Syntax

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

Inputs

Name	Type
value	FILETIME

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() [▶ 197].

Syntax

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

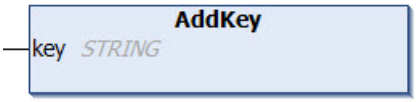
Inputs

Name	Type
pBytes	POINTER TO BYTE
nBytes	DINT

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 [▶ 194], AddBool [▶ 194], AddDateTime [▶ 195], AddDcTime [▶ 195], AddDint [▶ 195], AddFileTime [▶ 196], AddHexBinary [▶ 196], AddLint [▶ 201], AddLreal [▶ 202], AddNull [▶ 202], AddRawArray [▶ 202], AddRawObject [▶ 203], AddReal [▶ 203], AddString [▶ 204], AddUdint [▶ 204], AddUlint [▶ 204].

Syntax

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

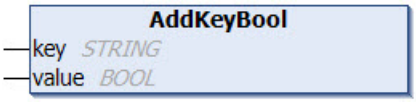
/ Inputs/Outputs

Name	Type
key	STRING

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
```

Inputs

Name	Type
value	BOOL

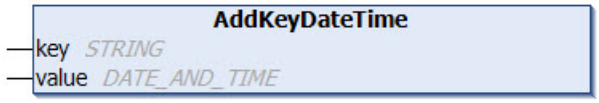
/ Inputs/Outputs

Name	Type
key	STRING

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
```

Inputs

Name	Type
value	DATE_AND_TIME

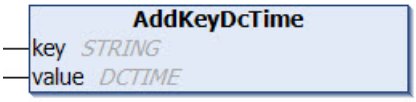
/ Inputs/Outputs

Name	Type
key	STRING

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
```

Inputs

Name	Type
value	DCTIME

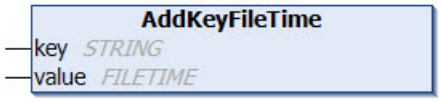
/ Inputs/Outputs

Name	Type
key	STRING

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
```

Inputs

Name	Type
value	FILETIME

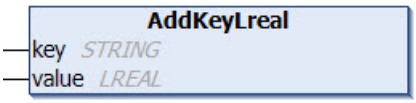
/ Inputs/Outputs

Name	Type
key	STRING

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
```

Inputs

Name	Type
value	LREAL

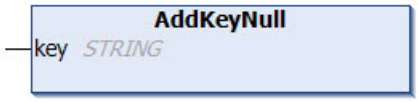
/ Inputs/Outputs

Name	Type
key	STRING

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 null.

Syntax

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

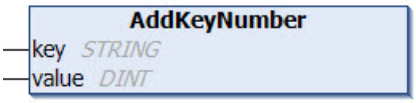
/ Inputs/Outputs

Name	Type
key	STRING

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
```

Inputs

Name	Type
value	DINT

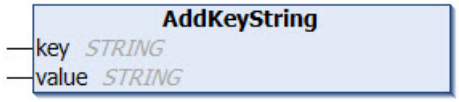
/ Inputs/Outputs

Name	Type
key	STRING

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
```

/ Inputs/Outputs

Name	Type
key	STRING
value	STRING

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() [▶ 197].

Syntax

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

Inputs

Name	Type
value	LINT

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() [▶ 197].

Syntax

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

Inputs

Name	Type
value	LREAL

Sample call:

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

5.2.1.4.19 AddNull

This method adds the value null to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 197].

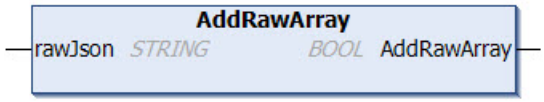
Syntax

```
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, e.g. directly after a preceding AddKey() [▶ 197], StartArray() [▶ 208] or as the first call after a ResetDocument() [▶ 208].

Syntax

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

Return value

Name	Type
AddRawArray	BOOL

Inputs/Outputs

Name	Type
rawJson	STRING

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 in an appropriately valid position, e.g. directly after a preceding AddKey() [▶ 197], StartArray() [▶ 208] or as the first call after a ResetDocument() [▶ 208].

Syntax

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

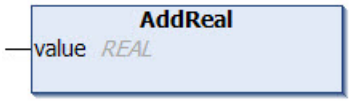
Inputs/Outputs

Name	Type
rawJson	STRING

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() [▶ 197].

Syntax

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

Inputs

Name	Type
value	REAL

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() [▶ 197].

Syntax

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

Inputs/Outputs

Name	Type
value	STRING

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() [▶ 197].

Syntax

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

Inputs

Name	Type
value	UDINT

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() [▶ 197].

Syntax

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

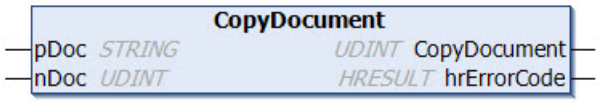
Inputs

Name	Type
value	ULINT

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 destination 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
```

Return value

Name	Type
CopyDocument	UDINT

Inputs

Name	Type
nDoc	UDINT

Inputs/Outputs

Name	Type
pDoc	STRING

Outputs

Name	Type
hrErrorCode	HRESULT

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
```

Return value

Name	Type
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
```

Return value

Name	Type
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 [▶ 205]() must be used.

Syntax

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

Return value

Name	Type
GetDocument	STRING(255)

Outputs

Name	Type
hrErrorCode	HRESULT

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
```

Return value

Name	Type
GetDocumentLength	UDINT

Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

5.2.1.4.31 GetMaxDecimalPlaces

This function queries the currently set number of decimal places after which a floating point number is truncated.

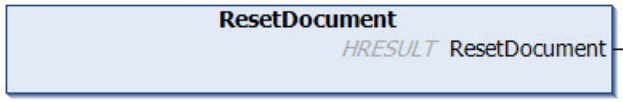
Syntax

```
METHOD GetMaxDecimalPlaces : DINT
```

Return value

Name	Type	Description
GetMaxDecimalPlaces	DINT	The currently set number of digits after which a floating point number is truncated.

5.2.1.4.32 ResetDocument



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

Syntax

```
METHOD ResetDocument : HRESULT
```

Return value

Name	Type
ResetDocument	HRESULT

Sample call:

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

5.2.1.4.33 SetMaxDecimalPlaces

This function determines the number of decimal places after which a floating-point number is truncated.

Syntax

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

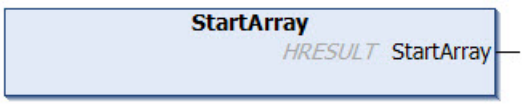
Return value

Name	Type
SetMaxDecimalPlaces	HRESULT

Inputs

Name	Type	Description
decimalPlaces	DINT	The number of decimal places to be set after which a floating-point number is truncated.

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
```

Return value

Name	Type
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
```

Return value

Name	Type
StartObject	HRESULT

Sample call:

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

5.2.1.5 FB_JsonSaxPrettyWriter

This function block has a difference from FB_JsonSaxWriter [▶ 191]: it adds matching whitespaces for better readability of a JSON document.

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.

Syntax

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

Outputs

Name	Type
initStatus	HRESULT

Methods

Name	Description
SetFormatOptions [▶ 224]	Adjusts the general formatting options for an instance of the [<=>] FB_JsonSaxPrettyWriter.
SetIndent [▶ 225]	Customizes the indentation.

All other methods can be found at FB_JsonSaxWriter [▶ 191].

Requirements

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

5.2.1.5.1 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() [▶ 213].

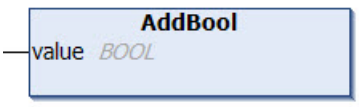
Syntax

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

Inputs

Name	Type
pBytes	POINTER TO BYTE
nBytes	DINT

5.2.1.5.2 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() [▶ 213].

Syntax

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

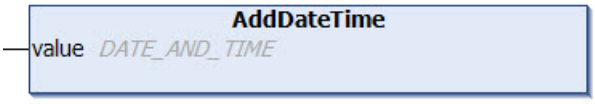
Inputs

Name	Type
value	BOOL

Sample call:

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

5.2.1.5.3 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() [▶ 213].

Syntax

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

Inputs

Name	Type
value	DATE_AND_TIME

Sample call:

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

5.2.1.5.4 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() [▶ 213].

Syntax

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

Inputs

Name	Type
value	DCTIME

Sample call:

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

5.2.1.5.5 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() [▶ 213].

Syntax

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

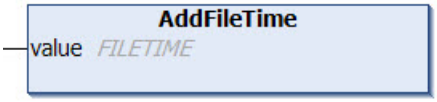
Inputs

Name	Type
value	DINT

Sample call:

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

5.2.1.5.6 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() [▶ 213].

Syntax

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

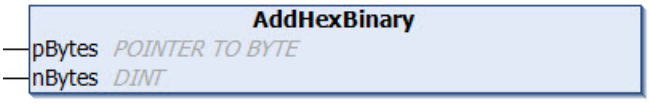
Inputs

Name	Type
value	FILETIME

Sample call:

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

5.2.1.5.7 AddHexBinary



This method adds a hex binary value to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 213].

Syntax

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

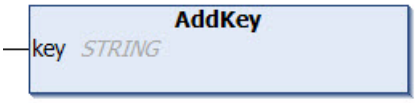
Inputs

Name	Type
pBytes	POINTER TO BYTE
nBytes	DINT

Sample call:

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

5.2.1.5.8 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 [▶ 210], AddBool [▶ 210], AddDateTime [▶ 211], AddDcTime [▶ 211], AddDint [▶ 212], AddFileTime [▶ 212], AddHexBinary [▶ 212], AddLint [▶ 217], AddLreal [▶ 218], AddNull [▶ 218], AddRawArray [▶ 218], AddRawObject [▶ 219], AddReal [▶ 219], AddString [▶ 220], AddUdint [▶ 220], AddUlint [▶ 221].

Syntax

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

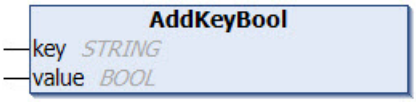
/ Inputs/Outputs

Name	Type
key	STRING

Sample call:

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

5.2.1.5.9 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
```

Inputs

Name	Type
value	BOOL

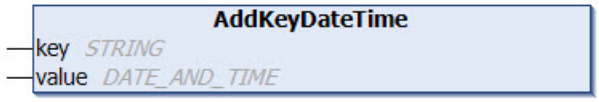
/ Inputs/Outputs

Name	Type
key	STRING

Sample call:

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

5.2.1.5.10 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
```

Inputs

Name	Type
value	DATE_AND_TIME

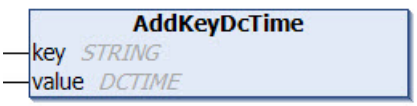
/ Inputs/Outputs

Name	Type
key	STRING

Sample call:

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

5.2.1.5.11 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
```

Inputs

Name	Type
value	DCTIME

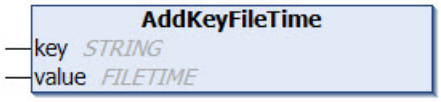
/ Inputs/Outputs

Name	Type
key	STRING

Sample call:

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

5.2.1.5.12 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
```

Inputs

Name	Type
value	FILETIME

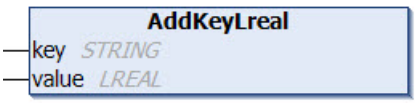
/ Inputs/Outputs

Name	Type
key	STRING

Sample call:

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

5.2.1.5.13 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
```

Inputs

Name	Type
value	LREAL

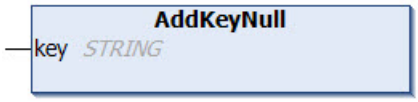
/ Inputs/Outputs

Name	Type
key	STRING

Sample call:

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

5.2.1.5.14 AddKeyNull



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

Syntax

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

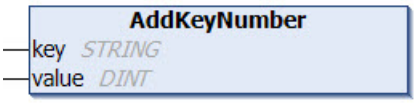
/ Inputs/Outputs

Name	Type
key	STRING

Sample call:

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

5.2.1.5.15 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
```

Inputs

Name	Type
value	DINT

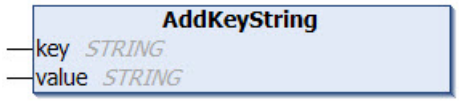
/ Inputs/Outputs

Name	Type
key	STRING

Sample call:

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

5.2.1.5.16 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
```

/ Inputs/Outputs

Name	Type
key	STRING
value	STRING

Sample call:

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

5.2.1.5.17 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() [▶ 213].

Syntax

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

Inputs

Name	Type
value	LINT

Sample call:

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

5.2.1.5.18 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() [▶ 213].

Syntax

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

Inputs

Name	Type
value	LREAL

Sample call:

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

5.2.1.5.19 AddNull

This method adds the value null to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 213].

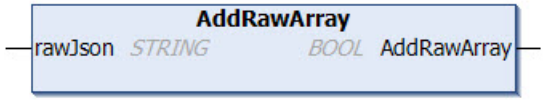
Syntax

```
METHOD AddNull
```

Sample call:

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

5.2.1.5.20 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, e.g. directly after a preceding AddKey() [▶ 213], StartArray() [▶ 226] or as the first call after a ResetDocument() [▶ 224].

Syntax

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

Return value

Name	Type
AddRawArray	BOOL

/ Inputs/Outputs

Name	Type
rawJson	STRING

Sample call:

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

5.2.1.5.21 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 in an appropriately valid position, e.g. directly after a preceding AddKey() [▶ 213], StartArray() [▶ 226] or as the first call after a ResetDocument() [▶ 224].

Syntax

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

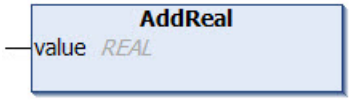
/ Inputs/Outputs

Name	Type
rawJson	STRING

Sample call:

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

5.2.1.5.22 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() [▶ 213].

Syntax

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

Inputs

Name	Type
value	REAL

Sample call:

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

5.2.1.5.23 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() [▶ 213].

Syntax

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

/ Inputs/Outputs

Name	Type
value	STRING

Sample call:

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

5.2.1.5.24 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() [▶ 213].

Syntax

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

Inputs

Name	Type
value	UDINT

Sample call:

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

5.2.1.5.25 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() [▶ 213].

Syntax

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

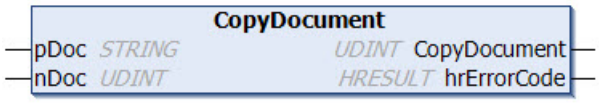
Inputs

Name	Type
value	ULINT

Sample call:

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

5.2.1.5.26 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 destination 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
```

Return value

Name	Type
CopyDocument	UDINT

Inputs

Name	Type
nDoc	UDINT

/ Inputs/Outputs

Name	Type
pDoc	STRING

Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

5.2.1.5.27 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
```

Return value

Name	Type
EndArray	HRESULT

Sample call:

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

5.2.1.5.28 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
```

Return value

Name	Type
EndObject	HRESULT

Sample call:

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

5.2.1.5.29 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 [▶ 221]() must be used.

Syntax

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

Return value

Name	Type
GetDocument	STRING(255)

Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

5.2.1.5.30 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
```

Return value

Name	Type
GetDocumentLength	UDINT

Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

5.2.1.5.31 GetMaxDecimalPlaces

This function queries the currently set number of decimal places after which a floating point number is truncated.

Syntax

```
METHOD GetMaxDecimalPlaces : DINT
```

Return value

Name	Type	Description
GetMaxDecimalPlaces	DINT	The currently set number of digits after which a floating point number is truncated.

5.2.1.5.32 ResetDocument

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

Syntax

```
METHOD ResetDocument : HRESULT
```

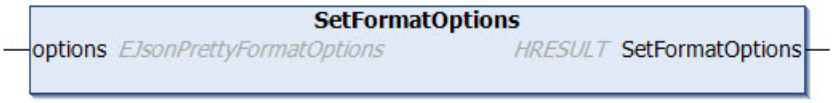
Return value

Name	Type
ResetDocument	HRESULT

Sample call:

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

5.2.1.5.33 SetFormatOptions



This method is used to customize the general formatting options for an instance of the FB_JsonSaxPrettyWriter [▶ 209]. In addition to the default configuration, there is also the possibility that no line breaks are generated within an array. The array is then output in one line despite the inserted indentations.

Syntax

```
METHOD SetFormatOptions : HRESULT
VAR_INPUT
  options               : EJsonPrettyFormatOptions;
END_VAR
TYPE EJsonPrettyFormatOptions:
(
  eFormatDefault        :=0,
  eFormatSingleLineArray:=1
) DINT;
```

Return value

Name	Type
SetFormatOptions	HRESULT

Inputs

Name	Type
options	EJsonPrettyFormatOptions

Sample call:

```
fbJson.SetFormatOptions(EJsonPrettyFormatOptions.eFormatSingleLineArray);
```

5.2.1.5.34 SetIndent

This method is used to customize the indentation.

Syntax

```
METHOD SetIndent : HRESULT
VAR_INPUT
  indentChar      : SINT;
  indentCharCount : UDINT;
END_VAR
```

Return value

Name	Type
SetIndent	HRESULT

Inputs

Name	Type
indentChar	SINT
indentCharCount	UDINT

Sample call:

```
fbJson.SetIndent(1,5);
```

5.2.1.5.35 SetMaxDecimalPlaces

This function determines the number of decimal places after which a floating-point number is truncated.

Syntax

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

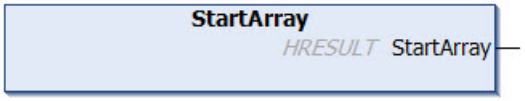
Return value

Name	Type
SetMaxDecimalPlaces	HRESULT

Inputs

Name	Type	Description
decimalPlaces	DINT	The number of decimal places to be set after which a floating-point number is truncated.

5.2.1.5.36 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
```

Return value

Name	Type
StartArray	HRESULT

Sample call:

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

5.2.1.5.37 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
```

Return value

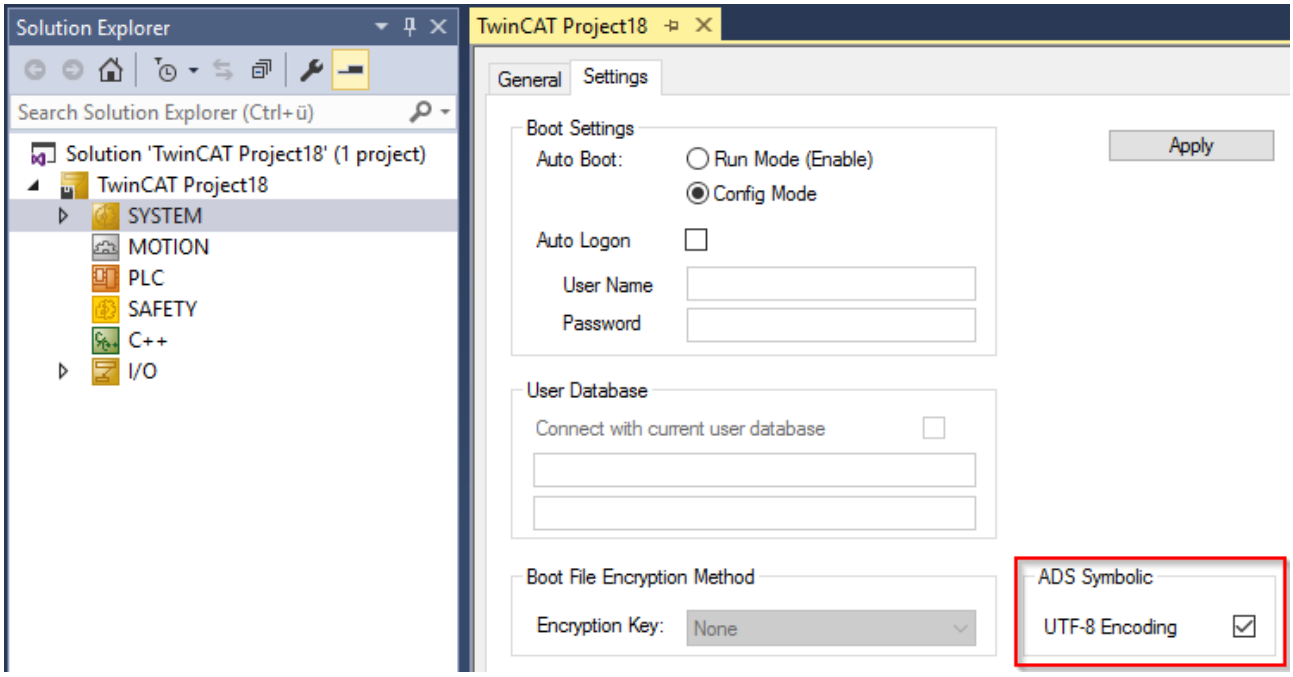
Name	Type
StartObject	HRESULT

Sample call:

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

5.2.1.6 FB_JsonReadWriteDataType

In order to use UTF-8 characters, e.g. in the automatic generation of metadata via the function block FB_JsonReadWriteDataType [▶ 226], 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_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.

Outputs

Name	Type
initStatus	HRESULT

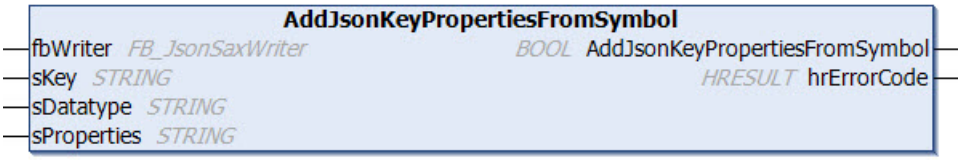
Methods

Name	Description
FB_JsonReadWriteDataType	Metadata can be added via PLC attributes to the JSON representation of a PLC data structure on an FB_JsonSaxWriter object.
AddJsonKeyValueFromSymbol [▶ 229]	Creates the JSON representation of a PLC data structure on an FB_JsonSaxWriter object.
AddJsonValueFromSymbol [▶ 230]	Creates the JSON representation of a PLC data structure on an FB_JsonSaxWriter object.
CopyJsonStringFromSymbol [▶ 231]	Creates the JSON representation of a symbol and copies it to a variable of data type STRING.
CopyJsonStringFromSymbolProperties [▶ 232]	Creates a corresponding JSON representation of PLC attributes on a symbol.
CopySymbolNameByAddress [▶ 233]	Returns the complete (ADS) symbol name of a transferred symbol.
GetDataTypeNameByAddress [▶ 234]	Returns the data type name of a transferred symbol.
GetJsonFromSymbol [▶ 235]	Creates the corresponding JSON representation of a symbol.
GetJsonStringFromSymbol [▶ 236]	Creates the corresponding JSON representation of a symbol.
GetJsonStringFromSymbolProperties [▶ 237]	Creates a corresponding JSON representation of PLC attributes on a symbol.
GetSizeJsonStringFromSymbol [▶ 238]	Reads the size of the JSON representation of a symbol.
GetSizeJsonStringFromSymbolProperties [▶ 239]	Reads the size of the JSON representation of PLC attributes on a symbol.
GetSymbolNameByAddress [▶ 239]	Returns the complete (ADS) symbol name of a transferred symbol.
SetSymbolFromJson [▶ 240]	Extracts a string containing a valid JSON message.

Requirements

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

5.2.1.6.1 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 [▶ 191] 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
```

Return value

Name	Type
AddJsonValueFromSymbol	BOOL

/ Inputs/Outputs

Name	Type
fbWriter	FB_JsonSaxWriter
sKey	STRING
sDatatype	STRING
sProperties	STRING

Outputs

Name	Type
hrErrorCode	HRESULT

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

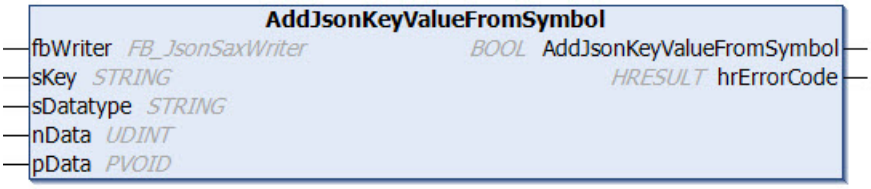
```
{attribute 'Unit'        := 'm/s'}
{attribute 'DisplayName' := 'Speed'}
Sensor1                  : REAL;
```

A complete sample of how to use this method can be found in section Tc3JsonXmlSampleJsonDataType [▶ 334].

Sample call:

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

5.2.1.6.2 AddJsonKeyValueFromSymbol



This method generates the JSON representation of a PLC data structure on an FB_JsonSaxWriter [▶ 191] 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() [▶ 230], 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
```

Return value

Name	Type
AddJsonValueFromSymbol	BOOL

Inputs

Name	Type
nData	UDINT
pData	PVOID

/ Inputs/Outputs

Name	Type
fbWriter	FB_JsonSaxWriter
sKey	STRING
sDatatype	STRING

Outputs

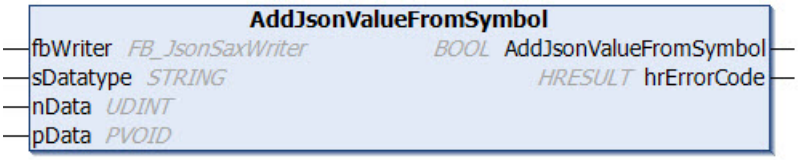
Name	Type
hrErrorCode	HRESULT

A complete sample of how to use this method can be found in section Tc3JsonXmlSampleJsonDataType [▶ 334].

Sample call:

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

5.2.1.6.3 AddJsonValueFromSymbol



This method generates the JSON representation of a PLC data structure on an FB_JsonSaxWriter [▶ 191] 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
```

Return value

Name	Type
AddJsonValueFromSymbol	BOOL

Inputs

Name	Type
nData	UDINT
pData	PVOID

Inputs/Outputs

Name	Type
fbWriter	FB_JsonSaxWriter
sDatatype	STRING

Outputs

Name	Type
hrErrorCode	HRESULT

A complete sample of how to use this method can be found in section Tc3JsonXmlSampleJsonDataType [▶ 334].

Sample call:

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

5.2.1.6.4 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 destination 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
```

Return value

Name	Type
CopyJsonStringFromSymbol	UDINT

Inputs

Name	Type
nData	UDINT
nDoc	UDINT
pData	PVOID

/ Inputs/Outputs

Name	Type
pDoc	STRING
sDatatype	STRING

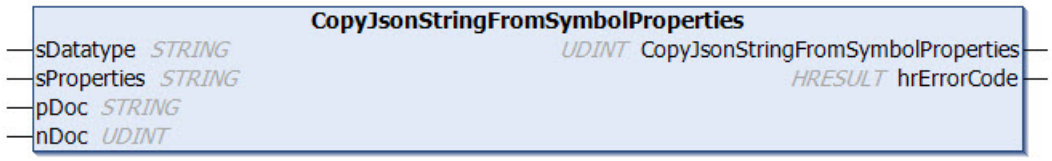
Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

5.2.1.6.5 CopyJsonStringFromSymbolProperties



This method generates a corresponding JSON representation of PLC attributes on a symbol. In contrast to the AddJsonKeyPropertiesFromSymbol [▶ 228] 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 destination 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
```

Return value

Name	Type
CopyJsonStringFromSymbolProperties	UDINT

Inputs

Name	Type
nDoc	UDINT

Inputs/Outputs

Name	Type
pDoc	STRING
sDatatype	STRING
sProperties	STRING

Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

5.2.1.6.6 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 destination 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
```

Return value

Name	Type
CopySymbolNameByAddress	UDINT

Inputs

Name	Type
nData	UDINT
pData	PVOID
nName	UDINT

Inputs/Outputs

Name	Type
sName	STRING

Outputs

Name	Type
hrErrorCode	HRESULT

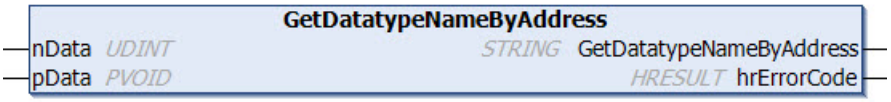
Sample call:

```
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

5.2.1.6.7 GetDataTypeNameByAddress



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

Structures with a single variable

If structures with a single variable are used, the data type cannot be read correctly using this method. The size and address information cannot be used to distinguish between the structure itself and the individual variables.

Syntax

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

Return value

Name	Type
GetDataTypeNameByAddress	STRING

Inputs

Name	Type
nData	UDINT
pData	PVOID

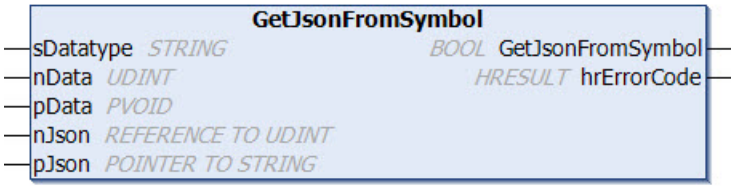
Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

5.2.1.6.8 GetJsonFromSymbol



This method generates the corresponding JSON representation of a symbol. In contrast to the AddJsonValueFromSymbol() [▶ 230] 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
```

Return value

Name	Type
GetJsonFromSymbol	BOOL

Inputs

Name	Type
nData	UDINT
pData	PVOID
nJson	REFERENCE TO UDINT
pJson	POINTER TO STRING

Inputs/Outputs

Name	Type
sDatatype	STRING

Outputs

Name	Type
hrErrorCode	HRESULT

Input parameter nJson

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

Sample call:

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

5.2.1.6.9 GetJsonStringFromSymbol

This method generates the corresponding JSON representation of a symbol. In contrast to the AddJsonValueFromSymbol() [▶ 230] 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 of a structure instance, for example.

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 [▶ 231]() 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
```

Return value

Name	Type
GetJsonStringFromSymbol	STRING(255)

Inputs

Name	Type
nData	UDINT
pData	PVOID

/ Inputs/Outputs

Name	Type
sDatatype	STRING

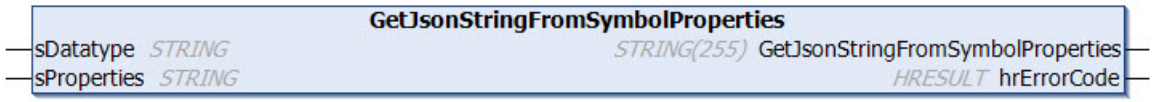
Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

5.2.1.6.10 GetJsonStringFromSymbolProperties



This method generates a corresponding JSON representation of PLC attributes on a symbol. In contrast to the AddJsonKeyPropertiesFromSymbol [▶ 228] 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 [▶ 232]() must be used.

Syntax

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

Return value

Name	Type
GetJsonStringFromSymbolProperties	STRING(255)

/ Inputs/Outputs

Name	Type
sDatatype	STRING
sProperties	STRING

Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

5.2.1.6.11 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
```

Return value

Name	Type
GetSizeJsonStringFromSymbol	UDINT

Inputs

Name	Type
nData	UDINT
pData	PVOID

/ Inputs/Outputs

Name	Type
sDatatype	STRING

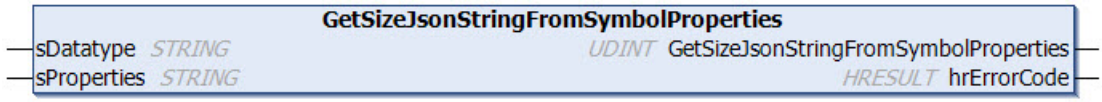
Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

5.2.1.6.12 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
```

Return value

Name	Type
GetSizeJsonStringFromSymbolProperties	UDINT

Inputs/Outputs

Name	Type
sDatatype	STRING
sProperties	STRING

Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

5.2.1.6.13 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() [▶ 233] must be used.

Syntax

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

Return value

Name	Type
GetSymbolNameByAddress	STRING(255)

Inputs

Name	Type
nData	UDINT
pData	PVOID

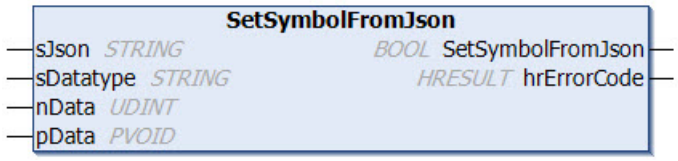
Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

5.2.1.6.14 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
```

Return value

Name	Type
SetSymbolFromJson	BOOL

Inputs

Name	Type
nData	UDINT
pData	PVOID

/ Inputs/Outputs

Name	Type
sJson	STRING
sDatatype	STRING

Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

5.2.1.7 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_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.

Outputs

Name	Type
initStatus	HRESULT

Methods

Name	Description
AppendAttribute [▶ 247]	Appends a new attribute to an existing node.
AppendAttributeAsBool [▶ 248]	Appends a new attribute to an existing node (Boolean).
AppendAttributeAsDouble [▶ 249]	Appends a new attribute to an existing node (Double).
AppendAttributeAsFloat [▶ 249]	Appends a new attribute to an existing node (Float).
AppendAttributeAsInt [▶ 250]	Appends a new attribute to an existing node (Integer).
AppendAttributeAsLint [▶ 251]	Appends a new attribute to an existing node (Integer64).
AppendAttributeAsUint [▶ 252]	Appends a new attribute to an existing node (Unsigned Integer).
AppendAttributeAsUlint [▶ 252]	Appends a new attribute to an existing node (Unsigned Integer64).
AppendAttributeCopy [▶ 253]	Appends a new attribute to an existing node.
AppendChild [▶ 254]	Appends a new node below an existing node.
AppendChildAsBool [▶ 254]	Appends a new node below an existing node (Boolean).
AppendChildAsDouble [▶ 255]	Appends a new node below an existing node (Double).
AppendChildAsFloat [▶ 256]	Appends a new node below an existing node (Float).
AppendChildAsInt [▶ 257]	Appends a new node below an existing node (Integer).
AppendChildAsLint [▶ 257]	Appends a new node below an existing node (Integer64).
AppendChildAsUint [▶ 258]	Appends a new node below an existing node (Unsigned Integer).
AppendChildAsUlint [▶ 259]	Appends a new node below an existing node (Unsigned Integer64).
AppendCopy [▶ 259]	Appends a new node below an existing node.
AppendNode [▶ 260]	Appends a new node below an existing node.
Attributes [▶ 261]	Reads the attribute of a given XML node.
AttributeAsBool [▶ 261]	Returns the value of an attribute as data type Boolean.
AttributeAsDouble [▶ 262]	Returns the value of an attribute as data type Double.
AttributeAsFloat [▶ 262]	Returns the value of an attribute as data type Float.
AttributeAsInt [▶ 263]	Returns the value of an attribute as data type Integer.
AttributeAsLint [▶ 263]	Returns the value of an attribute as data type Integer64.
AttributeAsUint [▶ 264]	Returns the value of an attribute as data type Unsigned Integer.
AttributeAsUlint [▶ 264]	Returns the value of an attribute as data type Unsigned Integer64.
AttributeBegin [▶ 265]	Returns an iterator over all attributes of an XML node.
AttributeFromIterator [▶ 265]	Converts the current position of an iterator to an XML attribute object.
AttributeName [▶ 266]	Returns the name of a given attribute.
Attributes [▶ 266]	Used to navigate the DOM and returns an iterator for all attributes found at an XML node.
AttributeText [▶ 267]	Returns the text of a given attribute.
Begin [▶ 268]	Returns an iterator over all child elements of an XML node.
BeginByName [▶ 268]	Returns an iterator over all child elements of an XML node, starting at a particular element.
Child [▶ 269]	Used to navigate through the DOM. It returns a reference to the (first) child element of the current node.
ChildByAttribute [▶ 269]	Used to navigate through the DOM. It returns a reference to a child element in the XML document.
ChildByAttributeAndName [▶ 270]	Used to navigate through the DOM. It returns a reference to a child element in the XML document.
ChildByName [▶ 271]	Used to navigate through the DOM. It returns a reference to a child element in the XML document.

Name	Description
Children [▶ 271]	Used to navigate through the DOM. It returns an iterator for several child elements found in the XML document.
ChildrenByName [▶ 272]	Used to navigate through the DOM. It returns an iterator for several child elements found in the XML document.
ClearIterator [▶ 273]	This method resets iterators that have already been used so that they can be used again the next time the program is run.
Compare [▶ 273]	Checks two iterators for equality.
CopyAttributeText [▶ 274]	Reads the value of an XML attribute and writes it to a variable of data type String.
CopyDocument [▶ 275]	Copies the contents of the DOM memory into a variable of data type String.
CopyNodeText [▶ 275]	Reads the value of an XML node and writes it to a variable of data type String.
CopyNodeXml [▶ 276]	Reads the XML structure of an XML node and writes it to a variable of data type String.
End	
FirstNodeByPath [▶ 277]	Navigates through an XML document using a path that was transferred to the method.
GetAttributeTextLength [▶ 277]	Returns the length of the value of an XML attribute.
GetDocumentLength [▶ 278]	Returns the length of an XML document in bytes.
GetDocumentNode [▶ 278]	Returns the root node of an XML document.
GetNodeTextLength [▶ 279]	Returns the length of the value of an XML node.
GetNodeXmlLength [▶ 279]	Returns the length of the XML structure of an XML node.
GetRootNode [▶ 280]	Returns a reference to the first XML node in the XML document.
InsertAttributeCopy [▶ 280]	Inserts an attribute to an XML node, copying the name and value of an existing attribute.
InsertAttribute [▶ 281]	Inserts an attribute to an XML node.
InsertChild [▶ 281]	Inserts a node to an existing XML node.
InsertCopy [▶ 282]	Inserts a new node to an existing XML node and copies an existing node.
IsEnd [▶ 283]	Checks whether a given XML iterator is at the end of the iteration that is to be performed.
LoadDocumentFromFile [▶ 283]	Loads a XML document from a file.
NewDocument [▶ 284]	Generates a new empty XML document in the DOM memory.
Next [▶ 284]	Sets an XML iterator for the next object that is to be processed.
NextAttribute [▶ 285]	Returns the next attribute for a given XML attribute.
NextByName [▶ 285]	Sets an XML iterator for the next object that is to be processed, which is identified by its name.
NextSibling [▶ 286]	Returns the next direct node for a given XML node at the same XML level.
NextSiblingByName [▶ 287]	Returns the next direct node for a given XML node with a particular name at the same XML level.
Node [▶ 287]	Used in conjunction with an iterator to navigate through the DOM.
NodeAsBool [▶ 288]	Returns the text of an XML node as data type Boolean.
NodeAsDouble [▶ 288]	Returns the text of an XML node as data type Double.
NodeAsFloat [▶ 289]	Returns the text of an XML node as data type Float.
NodeAsInt [▶ 289]	Returns the text of an XML node as data type Integer.
NodeAsLint [▶ 290]	Returns the text of an XML node as data type Integer64.
NodeAsUint [▶ 290]	Returns the text of an XML node as data type Unsigned Integer.
NodeAsUlint [▶ 291]	Returns the text of an XML node as data type Unsigned Integer64.

Name	Description
NodeName [▶ 291]	Returns the name of an XML node.
NodeText [▶ 292]	Returns the text of an XML node.
ParseDocument [▶ 292]	Loads an XML object into the DOM memory for further processing.
RemoveAttribute	
RemoveAttributeByName	
RemoveChild [▶ 293]	Removes an XML child node from a given XML node.
RemoveChildByName [▶ 293]	Removes an XML child node from a given XML node.
SaveDocumentToFile [▶ 294]	Saves the current XML document in a file.
SetAttribute [▶ 295]	Sets the value of an attribute.
SetAttributeAsBool [▶ 296]	Sets the value of an attribute (Boolean).
SetAttributeAsDouble [▶ 296]	Sets the value of an attribute (double).
SetAttributeAsFloat [▶ 297]	Sets the value of an attribute (float).
SetAttributeAsInt [▶ 297]	Sets the value of an attribute (integer).
SetAttributeAsLint [▶ 298]	Sets the value of an attribute (Integer64).
SetAttributeAsUint [▶ 298]	Sets the value of an attribute (Unsigned Integer).
SetAttributeAsUlint [▶ 299]	Sets the value of an attribute (Unsigned Integer64).
SetChild [▶ 299]	Sets the value of an XML node (String).
SetChildAsBool [▶ 300]	Sets the value of an XML node (Boolean).
SetChildAsDouble [▶ 300]	Sets the value of an XML node (Double).
SetChildAsFloat [▶ 301]	Sets the value of an XML node (Float).
SetChildAsInt [▶ 302]	Sets the value of an XML node (Integer).
SetChildAsLint [▶ 302]	Sets the value of an XML node (Integer64).
SetChildAsUint [▶ 303]	Sets the value of an XML node (Unsigned Integer).
SetChildAsUlint [▶ 303]	Sets the value of an XML node (Unsigned Integer64).

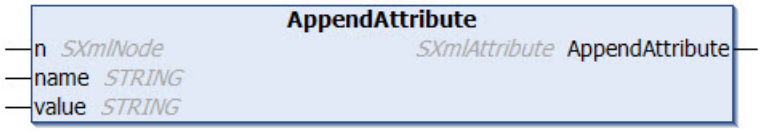
Requirements

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

Also see about this

- AppendAttribute [▶ 247]
- AppendAttributeAsBool [▶ 248]
- AppendAttributeAsDouble [▶ 249]
- AppendAttributeAsFloat [▶ 249]
- AppendAttributeAsInt [▶ 250]
- AppendAttributeAsLint [▶ 251]
- AppendAttributeAsUint [▶ 252]
- AppendAttributeAsUlint [▶ 252]
- AppendAttributeCopy [▶ 253]
- AppendChild [▶ 254]
- AppendChildAsBool [▶ 254]
- AppendChildAsDouble [▶ 255]
- AppendChildAsFloat [▶ 256]
- AppendChildAsInt [▶ 257]

5.2.1.7.1 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
```

Return value

Name	Type
AppendAttribute	SXmlAttribute

Inputs

Name	Type
n	SXmlNode

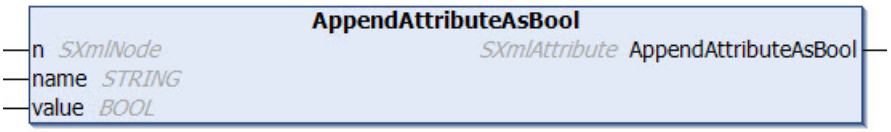
/ Inputs/Outputs

Name	Type
name	STRING
value	STRING

Sample call:

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

5.2.1.7.2 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
```

Return value

Name	Type
AppendAttributeAsBool	SXmlAttribute

Inputs

Name	Type
n	SXmlNode
value	BOOL

/ Inputs/Outputs

Name	Type
name	STRING

Sample call:

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

5.2.1.7.3 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
```

Return value

Name	Type
AppendAttributeAsDouble	SXmlAttribute

Inputs

Name	Type
n	SXmlNode
value	LREAL

/ Inputs/Outputs

Name	Type
name	STRING

Sample call:

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

5.2.1.7.4 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
```

Return value

Name	Type
AppendAttributeAsFloat	SXmlAttribute

Inputs

Name	Type
n	SXmlNode
value	REAL

/ Inputs/Outputs

Name	Type
name	STRING

Sample call:

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

5.2.1.7.5 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
```

Return value

Name	Type
AppendAttributeAsInt	SXmlAttribute

Inputs

Name	Type
n	SXmlNode
value	DINT

/ Inputs/Outputs

Name	Type
name	STRING

Sample call:

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

5.2.1.7.6 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
```

Return value

Name	Type
AppendAttributeAsLint	SXmlAttribute

Inputs

Name	Type
n	SXmlNode
value	LINT

/ Inputs/Outputs

Name	Type
name	STRING

Sample call:

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

5.2.1.7.7 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
```

Return value

Name	Type
AppendAttributeAsUint	SXmlAttribute

Inputs

Name	Type
n	SXmlNode
value	UDINT

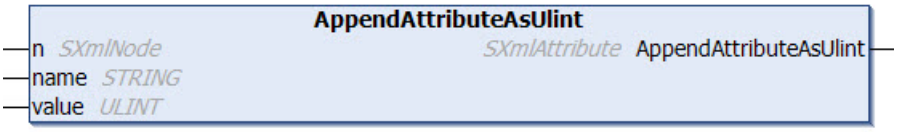
Inputs/Outputs

Name	Type
name	STRING

Sample call:

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

5.2.1.7.8 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
```

Return value

Name	Type
AppendAttributeAsUlint	SXmlAttribute

Inputs

Name	Type
n	SXmlNode
value	ULINT

/ Inputs/Outputs

Name	Type
name	STRING

Sample call:

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

5.2.1.7.9 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
```

Return value

Name	Type
AppendAttributeCopy	SXmlAttribute

Inputs

Name	Type
n	SXmlNode
copy	SXmlAttribute

Sample call:

```
xmlNewAttribute := fbXml.AppendAttributeCopy(xmlNode, xmlExistingAttribute);
```

5.2.1.7.10 AppendChild

This method inserts a new node below an existing node. The value of the new node is of the data type STRING. The name and the value of the new node as well as 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
```

Return value

Name	Type
AppendChild	SXmlNode

Inputs

Name	Type
n	SXmlNode
cdata	BOOL

/ Inputs/Outputs

Name	Type
name	STRING
value	STRING

Sample call:

```
xmlNewNode := fbXml.AppendChild(xmlExisting, 'Controller', 'CX5120', FALSE);
```

5.2.1.7.11 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
```

Return value

Name	Type
AppendChildAsBool	SXmlNode

Inputs

Name	Type
n	SXmlNode
value	BOOL

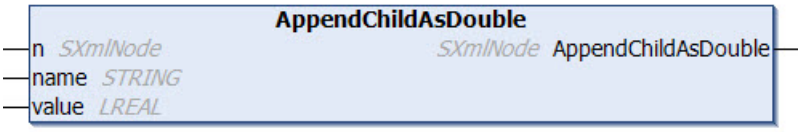
Inputs/Outputs

Name	Type
name	STRING

Sample call:

```
xmlNewNode := fbXml.AppendChildAsBool(xmlExisting, 'SomeName', TRUE);
```

5.2.1.7.12 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
```

Return value

Name	Type
AppendChildAsDouble	SXmlNode

Inputs

Name	Type
n	SXmlNode
value	LREAL

Inputs/Outputs

Name	Type
name	STRING

Sample call:

```
xmlNewNode := fbXml.AppendChildAsDouble(xmlExisting, 'SomeName', 42.42);
```

5.2.1.7.13 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
```

Return value

Name	Type
AppendChildAsFloat	SXmlNode

Inputs

Name	Type
n	SXmlNode
value	REAL

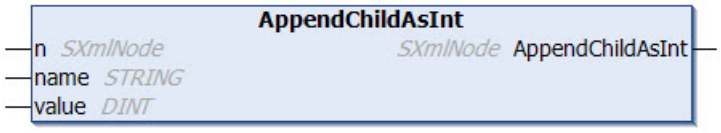
Inputs/Outputs

Name	Type
name	STRING

Sample call:

```
xmlNewNode := fbXml.AppendChildAsFloat(xmlExisting, 'SomeName', 42.42);
```

5.2.1.7.14 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
```

Return value

Name	Type
AppendChildAsInt	SXmlNode

Inputs

Name	Type
n	SXmlNode
value	DINT

/ Inputs/Outputs

Name	Type
name	STRING

Sample call:

```
xmlNewNode := fbXml.AppendChildAsInt(xmlExisting, 'SomeName', 42);
```

5.2.1.7.15 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
```

```
VAR_INPUT
  value : LINT;
END_VAR
```

Return value

Name	Type
AppendChildAsLint	SXmlNode

Inputs

Name	Type
n	SXmlNode
value	LINT

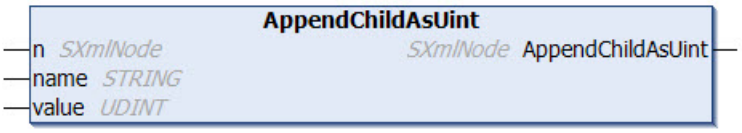
/ Inputs/Outputs

Name	Type
name	STRING

Sample call:

```
xmlNewNode := fbXml.AppendChildAsLint(xmlExisting, 'SomeName', 42);
```

5.2.1.7.16 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
```

Return value

Name	Type
AppendChildAsUint	SXmlNode

Inputs

Name	Type
n	SXmlNode
value	UDINT

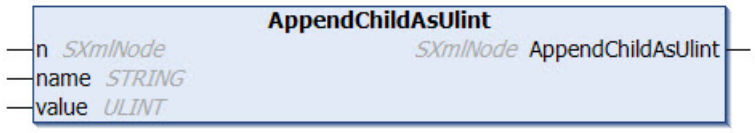
Inputs/Outputs

Name	Type
name	STRING

Sample call:

```
xmlNewNode := fbXml.AppendChildAsUint(xmlExisting, 'SomeName', 42);
```

5.2.1.7.17 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
```

Return value

Name	Type
AppendChildAsUlint	SXmlNode

Inputs

Name	Type
n	SXmlNode
value	ULINT

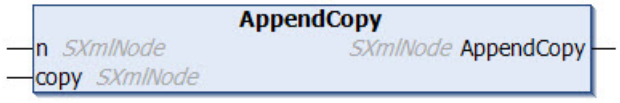
Inputs/Outputs

Name	Type
name	STRING

Sample call:

```
xmlNewNode := fbXml.AppendChildAsUlint(xmlExisting, 'SomeName', 42);
```

5.2.1.7.18 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
```

Return value

Name	Type
AppendCopy	SXmlNode

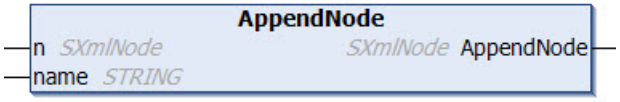
Inputs

Name	Type
n	SXmlNode
copy	SXmlNode

Sample call:

```
xmlNewNode := fbXml.AppendCopy(xmlParentNode, xmlExistingNode);
```

5.2.1.7.19 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
```

Return value

Name	Type
AppendNode	SXmlNode

Inputs

Name	Type
n	SXmlNode

Inputs/Outputs

Name	Type
name	STRING

Sample call:

```
objMachines := fbXml.AppendNode(objRoot, 'Machines');
```

5.2.1.7.20 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
```

Return value

Name	Type
Attributes	SXmlAttribute

Inputs

Name	Type
n	SXmlNode

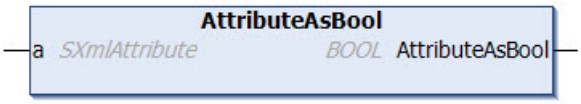
Inputs/Outputs

Name	Type
name	STRING

Sample call:

```
xmlMachine1Attribute := fbXml.Attribute(xmlMachine1, 'Type');
```

5.2.1.7.21 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
```

Return value

Name	Type
AttributeAsBool	BOOL

Inputs

Name	Type
a	SXmlAttribute

Sample call:

```
bValue := fbXml.AttributeAsBool(xmlAttr);
```

5.2.1.7.22 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
```

Return value

Name	Type
AttributeAsDouble	LREAL

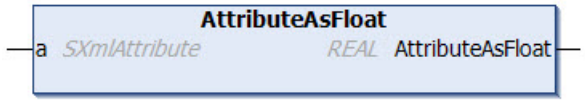
Inputs

Name	Type
a	SXmlAttribute

Sample call:

```
lrValue := fbXml.AttributeAsDouble(xmlAttr);
```

5.2.1.7.23 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
```

Return value

Name	Type
AttributeAsFloat	REAL

Inputs

Name	Type
a	SXmlAttribute

Sample call:

```
rValue := fbXml.AttributeAsFloat(xmlAttr);
```

5.2.1.7.24 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
```

Return value

Name	Type
AttributeAsInt	DINT

Inputs

Name	Type
a	SXmlAttribute

Sample call:

```
nValue := fbXml.AttributeAsInt(xmlAttr);
```

5.2.1.7.25 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
```

Return value

Name	Type
AttributeAsLint	LINT

Inputs

Name	Type
a	SXmlAttribute

Sample call:

```
nValue := fbXml.AttributeAsLint(xmlAttr);
```

5.2.1.7.26 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
```

Return value

Name	Type
AttributeAsUint	UDINT

Inputs

Name	Type
a	SXmlAttribute

Sample call:

```
nValue := fbXml.AttributeAsUint(xmlAttr);
```

5.2.1.7.27 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
```

Return value

Name	Type
AttributeAsUlint	ULINT

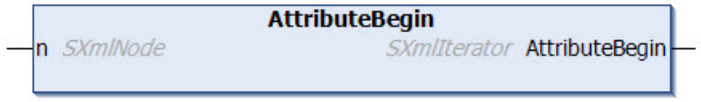
Inputs

Name	Type
a	SXmlAttribute

Sample call:

```
nValue := fbXml.AttributeAsUlint(xmlAttr);
```

5.2.1.7.28 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
```

Return value

Name	Type
AttributeBegin	SXmlIterator

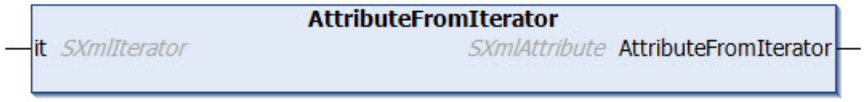
Inputs

Name	Type
n	SXmlNode

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
```

5.2.1.7.29 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
```

Return value

Name	Type
AttributeFromIterator	SXmlAttribute

Inputs

Name	Type
it	SXmlIterator

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
```

5.2.1.7.30 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
```

Return value

Name	Type
AttributeName	STRING

Inputs

Name	Type
a	SXmlAttribute

Sample call:

```
sName := fbXml.AttributeName(xmlAttr);
```

5.2.1.7.31 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
```

Return value

Name	Type
Attributes	SXmlAttribute

Inputs

Name	Type
it	REFERENCE TO SXmlIterator

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
```

5.2.1.7.32 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
```

Return value

Name	Type
AttributeText	STRING(255)

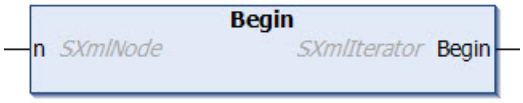
Inputs

Name	Type
a	SXmlAttribute

Sample call:

```
sText := fbXml.AttributeText(xmlAttr);
```

5.2.1.7.33 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
```

Return value

Name	Type
Begin	SXmlIterator

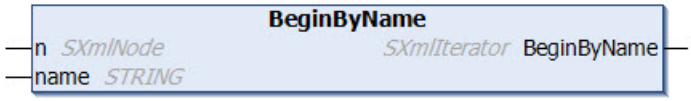
Inputs

Name	Type
n	SXmlNode

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
```

5.2.1.7.34 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
```

Return value

Name	Type
BeginByName	SXmlIterator

Inputs

Name	Type
n	SXmlNode

/ Inputs/Outputs

Name	Type
name	STRING

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
```

5.2.1.7.35 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
```

Return value

Name	Type

Inputs

Name	Type
n	SXmlNode

Sample call:

```
xmlChild := fbXml.Child(xmlNode);
```

5.2.1.7.36 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
```

Return value

Name	Type
ChildByAttribute	SXmlNode

Inputs

Name	Type
n	SXmlNode

/ Inputs/Outputs

Name	Type
attr	STRING
value	STRING

Sample call:

```
xmlMachine1 := fbXml.ChildByAttribute(xmlMachines, 'Type', '1');
```

5.2.1.7.37 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
```

Return value

Name	Type
ChildByAttributeAndName	SXmlNode

Inputs

Name	Type
n	SXmlNode

/ Inputs/Outputs

Name	Type
attr	STRING
value	STRING
child	STRING

Sample call:

```
xmlMachine2 := fbXml.ChildByAttributeAndName(xmlMachines, 'Type', '2', 'Machine');
```

5.2.1.7.38 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
```

Return value

Name	Type
ChildByName	SXmlNode

Inputs

Name	Type
n	SXmlNode

/ Inputs/Outputs

Name	Type
name	STRING

Sample call:

```
xmlMachines := fbXml.ChildByName(xmlDoc, 'Machines');
```

5.2.1.7.39 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
```

Return value

Name	Type
Children	SXmlNode

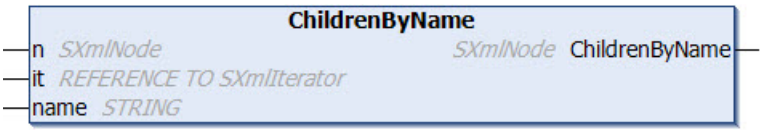
Inputs

Name	Type
n	SXmlNode

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
```

5.2.1.7.40 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
```

Return value

Name	Type
ChildrenByName	SXmlNode

Inputs

Name	Type
n	SXmlNode
it	REFERENCE TO SXmlIterator

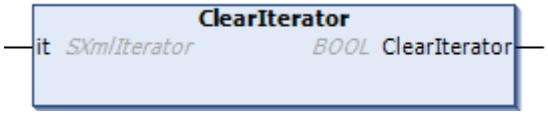
/ Inputs/Outputs

Name	Type
name	STRING

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
```

5.2.1.7.41 ClearIterator



This method resets iterators that have already been used so that they can be used again the next time the program is run.

Syntax

```
METHOD ClearIterator : BOOL
VAR_INPUT
  it : SXmlIterator;
END_VAR
```

Return value

Name	Type
ClearIterator	BOOL

Inputs

Name	Type
it	SXmlIterator

Sample call:

```
bResult := fbXml.ClearIterator(xmlIt);
```

Requirements

TwinCAT version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4024.56	x86, x64, ARM	Tc3_JsonXml 3.4.5.0

5.2.1.7.42 Compare

This method checks two iterators for equality.

Syntax

```
METHOD Compare : BOOL
VAR_INPUT
  it1 : SXmlIterator;
  it2 : SXmlIterator;
END_VAR
```

Return value

Name	Type
Compare	BOOL

Inputs

Name	Type
It1	SXmlIterator
It2	SXmlIterator

Sample call:

```
bResult := fbXml.Compare(xmlIt1, xmlIt2);
```

5.2.1.7.43 CopyAttributeText

This method reads the value of an XML attribute and writes it to a variable of data type String. The XML attribute as well as the target variable and the length to be written are passed 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
```

Return value

Name	Type
CopyAttributeText	UDINT

Inputs

Name	Type
a	SXmlAttribute
nXml	UDINT

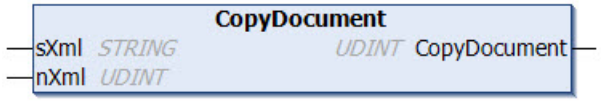
Inputs/Outputs

Name	Type
sXml	STRING

Sample call:

```
nLength := fbXml.CopyAttributeText(xmlAttr, sTarget, SIZEOF(sTarget));
```

5.2.1.7.44 CopyDocument



This method copies the contents of the DOM memory into a variable of data type String. The length to be written and the variable into which the resulting string is to be written are passed 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
```

Return value

Name	Type
CopyDocument	UDINT

Inputs

Name	Type
nXml	UDINT

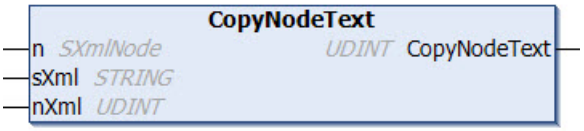
Inputs/Outputs

Name	Type
sXml	STRING

Sample call:

```
nLength := fbXml.CopyDocument(sTarget, SIZEOF(sTarget));
```

5.2.1.7.45 CopyNodeText



This method reads the value of an XML node and writes it to a variable of data type String. The XML node as well as the target variable and the length to be written are passed 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
```

Return value

Name	Type
CopyNodeText	UDINT

Inputs

Name	Type
n	SXmlNode
nXml	UDINT

Inputs/Outputs

Name	Type
sXml	STRING

Sample call:

```
nLength := fbXml.CopyNodeText(xmlNode, sTarget, SIZEOF(sTarget));
```

5.2.1.7.46 CopyNodeXml

This method reads the XML structure of an XML node and writes it to a variable of data type String. The XML node as well as the target variable and the length to be written are passed 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
```

Return value

Name	Type
CopyNodeXml	UDINT

Inputs

Name	Type
a	SXmlNode
nXml	UDINT

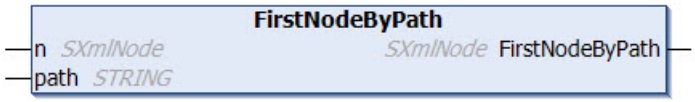
/ Inputs/Outputs

Name	Type
sXml	STRING

Sample call:

```
nLength := fbXml.CopyNodeXml(xmlNode, sTarget, SIZEOF(sTarget));
```

5.2.1.7.47 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
```

Return value

Name	Type
FirstNodeByPath	SXmlNode

Inputs

Name	Type
n	SXmlNode
path	STRING

Sample call:

```
xmlFoundNode := fbXml.FirstNodeByPath(xmlStartNode, 'Level1/Level2/Level3');
```

5.2.1.7.48 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
```

Return value

Name	Type
GetAttributeTextLength	UDINT

Inputs

Name	Type
a	SXmlAttribute

Sample call:

```
nLength := fbXml.GetAttributeTextLength(xmlAttr);
```

5.2.1.7.49 GetDocumentLength

This method returns the length of an XML document in bytes.

Syntax

```
METHOD GetDocumentLength : UDINT
```

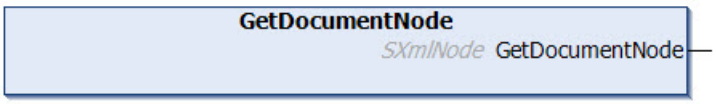
Return value

Name	Type
GetDocumentLength	UDINT

Sample call:

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

5.2.1.7.50 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
```

Return value

Name	Type
GetDocumentNode	SXmlNode

Sample call:

```
objRoot := fbXml.GetDocumentNode();
```

5.2.1.7.51 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
```

Return value

Name	Type
GetNodeTextLength	UDINT

Inputs

Name	Type
n	SXmlNode

Sample call:

```
nLength := fbXml.GetNodeTextLength(xmlNode);
```

5.2.1.7.52 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
```

Return value

Name	Type
GetNodeXmlLength	UDINT

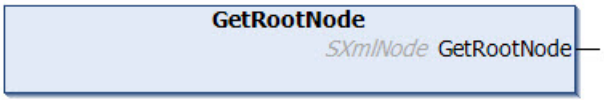
Inputs

Name	Type
n	SXmlNode

Sample call:

```
nLength := fbXml.GetNodeXmlLength(xmlNode);
```

5.2.1.7.53 GetRootNode



This method returns a reference to the first XML node in the XML document.

Syntax

```
METHOD GetRootNode : SXmlNode
```

Return value

Name	Type
GetRootNode	SXmlNode

Sample call:

```
xmlRootNode := fbXml.GetRootNode();
```

5.2.1.7.54 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
```

Return value

Name	Type
InsertAttributeCopy	SXmlAttribute

Inputs

Name	Type
n	SXmlNode
before	SXmlAttribute
copy	SXmlAttribute

Sample call:

```
xmlNewAttr := fbXml.InsertAttributeCopy(xmlNode, xmlBeforeAttr, xmlCopyAttr);
```

5.2.1.7.55 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
```

Return value

Name	Type
InsertAttribute	SXmlAttribute

Inputs

Name	Type
n	SXmlNode
before	SXmlAttribute

Inputs/Outputs

Name	Type
name	STRING

Sample call:

```
xmlNewAttr := fbXml.InsertAttribute(xmlNode, xmlBeforeAttr, 'SomeName');
```

5.2.1.7.56 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
```

Return value

Name	Type
InsertChild	SXmlNode

Inputs

Name	Type
n	SXmlNode
before	SXmlAttribute

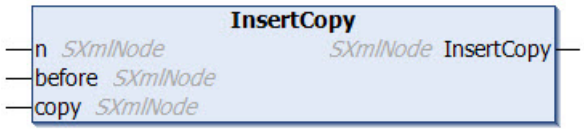
/ Inputs/Outputs

Name	Type
name	STRING

Sample call:

```
xmlNewNode := fbXml.InsertChild(xmlNode, xmlBeforeNode, 'SomeName');
```

5.2.1.7.57 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
```

Return value

Name	Type
InsertCopy	SXmlNode

Inputs

Name	Type
n	SXmlNode
before	SXmlNode
copy	SXmlNode

Sample call:

```
xmlNewNode := fbXml.InsertCopy(xmlNode, xmlBeforeNode, xmlCopyNode);
```

5.2.1.7.58 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
```

Return value

Name	Type
IsEnd	BOOL

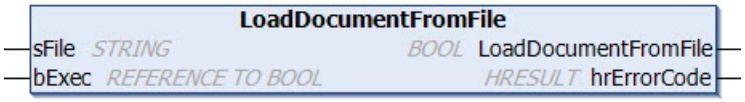
Inputs

Name	Type
nit	SXmlIterator

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
```

5.2.1.7.59 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 whether the loading of the file was successful (TRUE) or failed (FALSE) for the duration of one call.

Encoding

Please note that the imported files must be formatted in UTF-8. Formats such as UTF-16 are not supported.

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
```

Return value

Name	Type
LoadDocumentFromFile	BOOL

Inputs

Name	Type
bExec	REFERENCE TO BOOL

/ Inputs/outputs

Name	Type
sFile	STRING

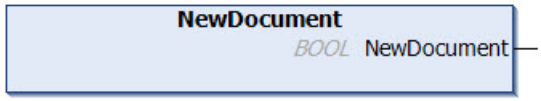
Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

```
IF bLoad THEN
  bLoaded := fbXml.LoadDocumentFromFile('C:\Test.xml', bLoad);
END_IF
```

5.2.1.7.60 NewDocument



This method creates an empty XML document in the DOM memory.

Syntax

```
METHOD NewDocument : BOOL
```

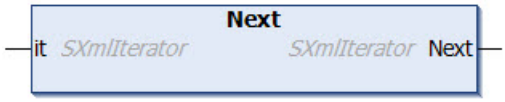
Return value

Name	Type
NewDocument	BOOL

Sample call:

```
fbXml.NewDocument();
```

5.2.1.7.61 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
```

Return value

Name	Type
Next	SXmlIterator

Inputs

Name	Type
it	SXmlIterator

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
```

5.2.1.7.62 NextAttribute

This method returns the next attribute for a given XML attribute.

Syntax

```
METHOD NextAttribute : SXmlAttribute
VAR_INPUT
  a : SXmlAttribute;
END_VAR
```

Return value

Name	Type
NextAttribute	SXmlAttribute

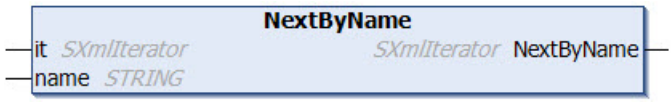
Inputs

Name	Type
a	SXmlAttribute

Sample call:

```
xmlNextAttr := fbXml.NextAttribute(xmlAttr);
```

5.2.1.7.63 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
```

Return value

Name	Type
NextByName	SXmlIterator

Inputs

Name	Type
it	SXmlIterator

/ Inputs/Outputs

Name	Type
name	STRING

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
```

5.2.1.7.64 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
```

Return value

Name	Type
NextSibling	SXmlNode

Inputs

Name	Type
n	SXmlNode

Sample call:

```
xmlSibling := fbXml.NextSibling(xmlNode);
```

5.2.1.7.65 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
```

Return value

Name	Type
NextSiblingByName	SXmlNode

Inputs

Name	Type
n	SXmlNode

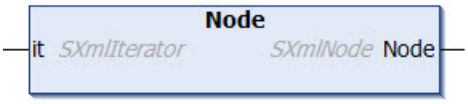
/ Inputs/Outputs

Name	Type
name	STRING

Sample call:

```
xmlSibling := fbXml.NextSiblingByName(xmlNode, 'SomeName');
```

5.2.1.7.66 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
```

Return value

Name	Type
Node	SXmlNode

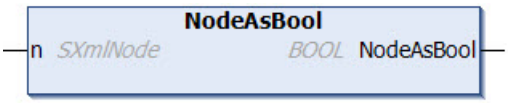
Inputs

Name	Type
it	SXmlIterator

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
```

5.2.1.7.67 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
```

Return value

Name	Type
NodeAsBool	BOOL

Inputs

Name	Type
n	SXmlNode

Sample call:

```
bXmlNode:= fbXml.NodeAsBool(xmlMachine1);
```

5.2.1.7.68 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
```

Return value

Name	Type
NodeAsDouble	LREAL

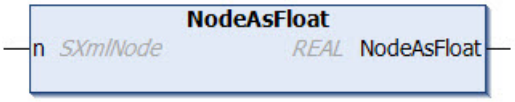
Inputs

Name	Type
n	SXmlNode

Sample call:

```
lrXmlNode:= fbXml.NodeAsDouble(xmlMachine1);
```

5.2.1.7.69 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
```

Return value

Name	Type
NodeAsFloat	REAL

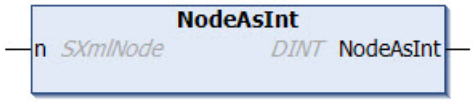
Inputs

Name	Type
n	SXmlNode

Sample call:

```
rXmlNode:= fbXml.NodeAsFloat(xmlMachine1);
```

5.2.1.7.70 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
```

Return value

Name	Type
NodeAsInt	DINT

Inputs

Name	Type
n	SXmlNode

Sample call:

```
nXmlNode:= fbXml.NodeAsInt(xmlMachine1);
```

5.2.1.7.71 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
```

Return value

Name	Type
NodeAsLint	LINT

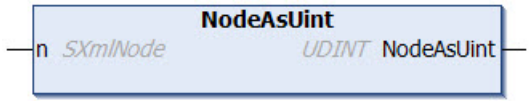
Inputs

Name	Type
n	SXmlNode

Sample call:

```
nXmlNode:= fbXml.NodeAsLint(xmlMachine1);
```

5.2.1.7.72 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
```

Return value

Name	Type
NodeAsUint	UDINT

Inputs

Name	Type
n	SXmlNode

Sample call:

```
nXmlNode:= fbXml.NodeAsUint(xmlMachine1);
```

5.2.1.7.73 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
```

Return value

Name	Type
NodeAsUlint	ULINT

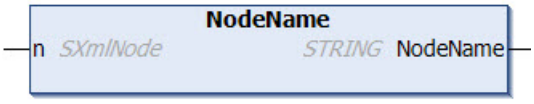
Inputs

Name	Type
n	SXmlNode

Sample call:

```
nXmlNode:= fbXml.NodeAsUlint(xmlMachine1);
```

5.2.1.7.74 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
```

Return value

Name	Type
NodeName	STRING

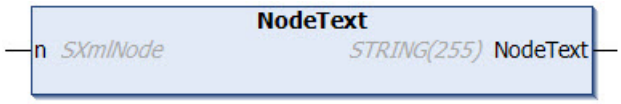
Inputs

Name	Type
n	SXmlNode

Sample call:

```
sNodeName := fbXml.NodeName(xmlMachine1);
```

5.2.1.7.75 NodeText



This method returns the text of an XML node. The XML node is transferred to the method as input parameter.

If the text of an XML node is longer than 255 characters, the method CopyNodeText [▶ 275] must be used.

Syntax

```
METHOD NodeText : STRING(255)
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Return value

Name	Type
NodeText	STRING(255)

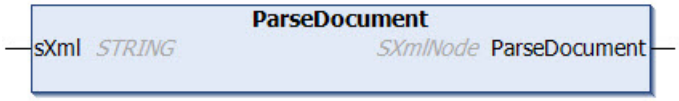
Inputs

Name	Type
n	SXmlNode

Sample call:

```
sMachine1Name := fbXml.NodeText(xmlMachine1);
```

5.2.1.7.76 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
```

Return value

Name	Type
ParseDocument	SXmlNode

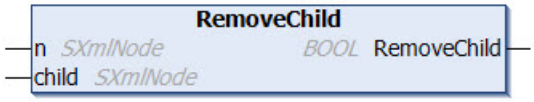
Inputs/Outputs

Name	Type
sXml	STRING

Sample call:

```
xmlDoc := fbXml.ParseDocument(sXmlToParse);
```

5.2.1.7.77 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
```

Return value

Name	Type
RemoveChild	BOOL

Inputs

Name	Type
n	SXmlNode
child	SXmlNode

Sample call:

```
bRemoved := fbXml.RemoveChild(xmlParent, xmlChild);
```

5.2.1.7.78 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
```

Return value

Name	Type
RemoveChildByName	BOOL

Inputs

Name	Type
n	SXmlNode

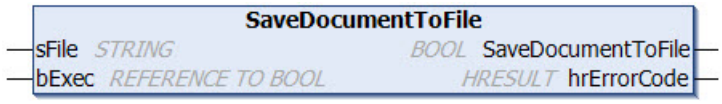
/ Inputs/Outputs

Name	Type
name	STRING

Sample call:

```
bRemoved := fbXml.RemoveChildByName(xmlParent, 'SomeName');
```

5.2.1.7.79 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
```

Return value

Name	Type
SaveDocumentToFile	BOOL

Inputs

Name	Type
bExec	REFERENCE TO BOOL

/ Inputs/Outputs

Name	Type
sFile	STRING

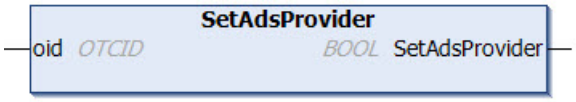
Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

```
IF bSave THEN
  bSaved = fbXml.SaveDocumentToFile('C:\Test.xml', bSave);
END_IF
```

5.2.1.7.80 SetAdsProvider



Syntax

```
METHOD SetAdsProvider : BOOL
VAR_IN_OUT CONSTANT
  oid : OTCID;
END_VAR
```

Return value

Name	Type
SetAdsProvider	BOOL

Inputs/Outputs

Name	Type
oid	OTCID

5.2.1.7.81 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
```

Return value

Name	Type
SetAttribute	SXmlAttribute

Inputs

Name	Type
a	SXmlAttribute

/ Inputs/Outputs

Name	Type
value	STRING

Sample call:

```
xmlAttr := fbXml.SetAttribute(xmlExistingAttr, 'Test');
```

5.2.1.7.82 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
```

Return value

Name	Type
SetAttributeAsBool	SXmlAttribute

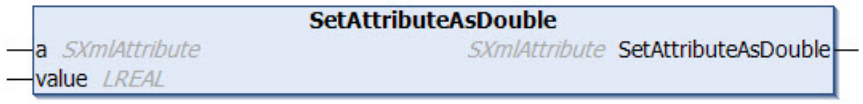
Inputs

Name	Type
a	SXmlAttribute
value	BOOL

Sample call:

```
xmlAttr := fbXml.SetAttributeAsBool(xmlExistingAttr, TRUE);
```

5.2.1.7.83 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
```

Return value

Name	Type
SetAttributeAsDouble	SXmlAttribute

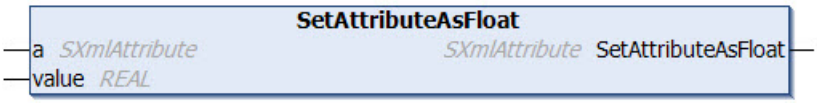
Inputs

Name	Type
a	SXmlAttribute
value	LREAL

Sample call:

```
xmlAttr := fbXml.SetAttributeAsDouble(xmlExistingAttr, 42.42);
```

5.2.1.7.84 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
```

Return value

Name	Type
SetAttributeAsFloat	SXmlAttribute

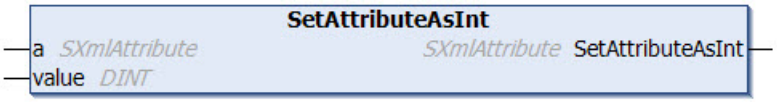
Inputs

Name	Type
a	SXmlAttribute
value	REAL

Sample call:

```
xmlAttr := fbXml.SetAttributeAsFloat(xmlExistingAttr, 42.42);
```

5.2.1.7.85 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
```

Return value

Name	Type
SetAttributeAsInt	SXmlAttribute

Inputs

Name	Type
a	SXmlAttribute
value	DINT

Sample call:

```
xmlAttr := fbXml.SetAttributeAsInt(xmlExistingAttr, 42);
```

5.2.1.7.86 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
```

Return value

Name	Type
SetAttributeAsLint	SXmlAttribute

Inputs

Name	Type
a	SXmlAttribute
value	LINT

Sample call:

```
xmlAttr := fbXml.SetAttributeAsLint(xmlExistingAttr, 42);
```

5.2.1.7.87 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
```

Return value

Name	Type
SetAttributeAsUint	SXmlAttribute

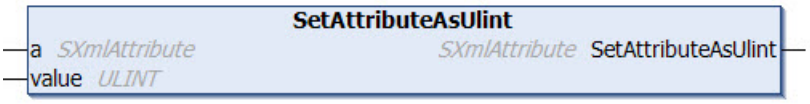
Inputs

Name	Type
a	SXmlAttribute
value	UDINT

Sample call:

```
xmlAttr := fbXml.SetAttributeAsUint(xmlExistingAttr, 42);
```

5.2.1.7.88 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
```

Return value

Name	Type
SetAttributeAsUlint	SXmlAttribute

Inputs

Name	Type
a	SXmlAttribute
value	ULINT

Sample call:

```
xmlAttr := fbXml.SetAttributeAsUlint(xmlExistingAttr, 42);
```

5.2.1.7.89 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
```

Return value

Name	Type
SetChild	SXmlNode

Inputs

Name	Type
n	SXmlNode
cdata	BOOL

/ Inputs/Outputs

Name	Type
value	STRING

Sample call:

```
xmlNode := fbXml.SetChild(xmlExistingNode, 'SomeText', FALSE);
```

5.2.1.7.90 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
```

Return value

Name	Type
SetChildAsBool	SXmlNode

Inputs

Name	Type
n	SXmlNode
value	BOOL

Sample call:

```
xmlNode := fbXml.SetChild(xmlExistingNode, TRUE);
```

5.2.1.7.91 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
```

Return value

Name	Type
SetChildAsDouble	SXmlNode

Inputs

Name	Type
n	SXmlNode
value	LREAL

Sample call:

```
xmlNode := fbXml.SetChildAsDouble(xmlExistingNode, 42.42);
```

5.2.1.7.92 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
```

Return value

Name	Type
SetChildAsFloat	SXmlNode

Inputs

Name	Type
n	SXmlNode
value	REAL

Sample call:

```
xmlNode := fbXml.SetChildAsFloat(xmlExistingNode, 42.42);
```

5.2.1.7.93 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
```

Return value

Name	Type
SetChildAsInt	SXmlNode

Inputs

Name	Type
n	SXmlNode
value	DINT

Sample call:

```
xmlNode := fbXml.SetChildAsInt(xmlExistingNode, 42);
```

5.2.1.7.94 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
```

Return value

Name	Type
SetChildAsLint	SXmlNode

Inputs

Name	Type
n	SXmlNode
value	LINT

Sample call:

```
xmlNode := fbXml.SetChildAsLint(xmlExistingNode, 42);
```

5.2.1.7.95 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
```

Return value

Name	Type
SetChildAsUint	SXmlNode

Inputs

Name	Type
n	SXmlNode
value	UDINT

Sample call:

```
xmlNode := fbXml.SetChildAsUint(xmlExistingNode, 42);
```

5.2.1.7.96 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
```

Return value

Name	Type
SetChildAsUlint	SXmlNode

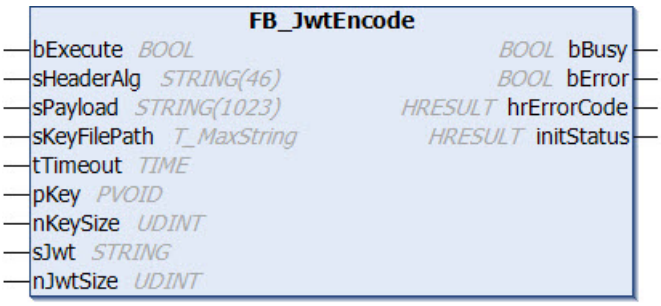
Inputs

Name	Type
n	SXmlNode
value	ULINT

Sample call:

```
xmlNode := fbXml.SetChildAsUlint(xmlExistingNode, 42);
```

5.2.1.8 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 [▶ 62]	Contains the fully coded and signed JWT after the function block has been processed.
nJwtSize	UDINT [▶ 63]	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 [▶ 339] 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.

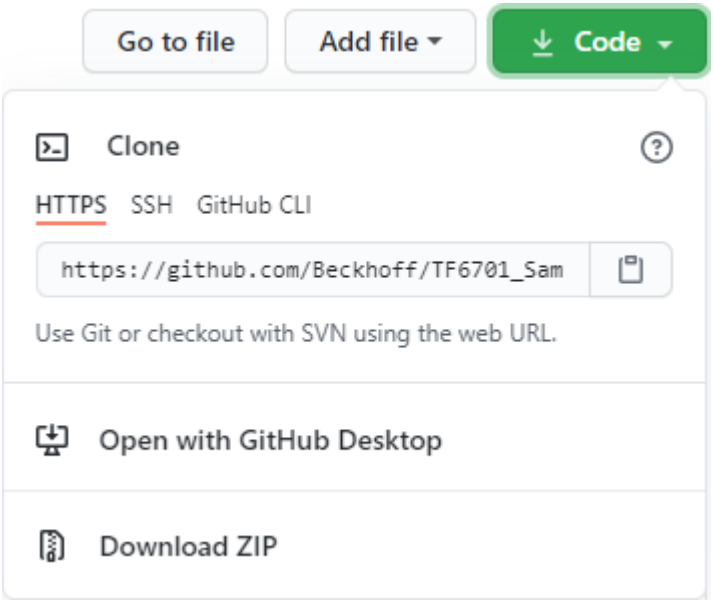
There are two different implementation options, which can be chosen based on purely subjective criteria. The two options are compared in the first two samples.

Overview

Sample	Link	Description
1	IotMqttSampleUsingQueue [▶ 312]	MQTT communication based on a messages queue
2	IotMqttSampleUsingCallback [▶ 314]	MQTT communication based on a callback method
3	IotMqttSampleTlsPsk [▶ 316]	MQTT communication via a secure TLS connection and PSK (PreSharedKey)
4	IotMqttSampleTlsCa [▶ 317]	MQTT communication via a secure TLS connection and CA (certificate authority) certificate
5	IotMqttSampleAwsIoT [▶ 317]	MQTT communication with AWS IoT
6	IotMqttSampleAzureIotHub [▶ 320]	MQTT communication with the Microsoft Azure IoT Hub
7	IotMqttSampleIbmWatsonIoT [▶ 323]	MQTT communication with IBM Watson IoT
8	IotMqttSampleMathworksThingspeak [▶ 324]	MQTT communication with the ThingSpeak IoT platform from MathWorks
9	IotMqttv5Sample [▶ 325]	MQTTv5 communication based on a messages queue
10	IotMqttv5LastWillSample [▶ 328]	Demonstrates the use of LastWill in interaction with the MQTTv5 function blocks.
11	IotMqttv5ReqResSample [▶ 328]	Demonstrates the request/response mechanism of MQTTv5, as well as the handling of UserProperties and CorrelationData
12	IotMqttv5UserPropsSample [▶ 329]	Demonstrates the use of user properties, which are part of MQTTv5.
13	JsonXmlSamples [▶ 330]	Contains various samples demonstrating the use of the JSON/XML parsers from the PLC library Tc3_JsonXml.

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 IotMqttSampleUsingQueue

Sample for MQTT communication via a message queue

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"). 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_IotBase as library reference.
2. Create a program block and declare an instance of FB_IotMqttClient [▶ 55] 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 message reception, declare a variable that contains the topic to be received and two other variables that indicate the topic and payload of the last received message.
   The received messages are to be collected in a queue so that they can be evaluated one after the other. For this you declare an instance of FB_IotMqttMessageQueue [▶ 64] and an instance of FB_IotMqttMessage [▶ 66].

```
    (* 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. Likewise, a message is to be sent every second.
   In the sample `sTopicPub = sTopicSub`, so that a loopback is created. In other applications the topics usually differ.

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

7. 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), bS
etNullTermination:=FALSE);
        ELSIF fbMessage.CompareTopic(sTopic:=sTopicSub2) THEN
            fbMessage.GetPayload(pPayload:=ADR(sPayloadRcv2), nPayloadSize:=SIZEOF(sPayloadRcv2), bS
etNullTermination:=FALSE);
        END_IF
    END_IF
END_IF
```

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.0	IPC or CX (x86, x64, ARM)	Tc3_IotBase, Tc2_Utilities (>= v3.3.19.0)

6.2 IotMqttSampleUsingCallback

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_IotMqttClient.Execute() method.

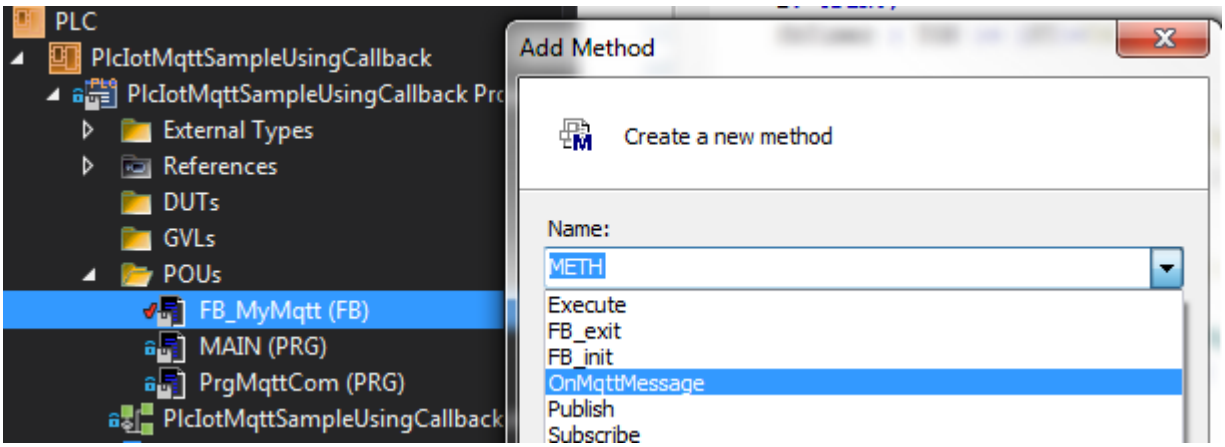
Project structure

1. Create a TwinCAT project with a PLC and add Tc3_IotBase 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_IotMqttClient 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 [▶ 312]. 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.AtMostOnceDe
livery);
    END_IF
    fbTimer(IN:=TRUE);
    IF fbTimer.Q THEN // publish new payload every second
        fbTimer(IN:=FALSE);
        i := i + 1;
        sPayloadPub := CONCAT('MyMessage', TO_STRING(i));
        fbMqttClient.Publish(    sTopic:= sTopicPub,
                                 pPayload:= ADR(sPayloadPub), nPayloadSize:= LEN2(ADR(sPayloadPub
))+1,
                                 eQoS:= TcIotMqttQos.AtMostOnceDelivery, bRetain:= FALSE, bQueue:
= FALSE );
    END_IF
END_IF
```

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.0	IPC or CX (x86, x64, ARM)	Tc3_IotBase, Tc2_Utilities (>= v3.3.19.0)

6.3 IotMqttSampleTlsPsk

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_IotBase as library reference.
2. Create a program block and declare an instance of FB_IotMqttClient [▶ 55] 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_IotBase, Tc2_Utilities (>= v3.3.19.0)

6.4 IotMqttSampleTlsCa

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 IotMqttSampleUsingQueue [▶ 312] and in particular IotMqttSampleAwsIoT [▶ 317]. 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_IotBase, Tc2_Utilities (>= v3.3.19.0)

6.5 IotMqttSampleAwsIoT

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. You can use self-signed certificates on the AWS IoT Management Console or have certificates signed by your own Certificate Authority (CA). In the latter case, your own CA must be trusted accordingly by the AWS IoT Core platform.

Initial setup of AWS IoT Core

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

The topic structure of the AWS IoT Core message broker is essentially freely selectable, although some restrictions apply. There are certain system topics that may not 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

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

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 IotMqttSampleUsingQueue [▶ 312], 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	Amazon Root CA certificate, which can be downloaded from the AWS IoT Management Console.
fbMqttClient.stTLS.sCert	Certificate of the device, which is required for authentication to establish a connection to AWS IoT. Both self-signed certificates and CA-signed certificates can be used. In the case of CA-signed certificates, the own CA must be classified as trustworthy on the AWS IoT Core Management Console. For more information, please refer to the AWS IoT Core documentation.
fbMqttClient.stTLS.sKeyFile	Private key of the device.
fbMqttClient.sHostname	Host name 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\AmazonRootCA1.pem';
  fbMqttClient.stTLS.sCert := 'c:\TwinCAT\3.1\Config\Certificates\7613eee18a-certificate.pem.crt';
  fbMqttClient.stTLS.sKeyFile := 'c:\TwinCAT\3.1\Config\Certificates\7613eee18a-private.pem.key';
  fbMqttClient.sHostName:= 'a35raby201xp77.iot.eu-west-1.amazonaws.com';
  fbMqttClient.nHostPort:= 8883;
  fbMqttClient.sClientId:= 'CX-12345';
  fbMqttClient.ipMessageQueue := fbMessageQueue;
  fbMqttClient.ActivateExponentialBackoff(T#1S, T#30S);
END_IF
```

Exponential backoff

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() [▶ 61] 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() [▶ 62] 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_IotBase, Tc2_Utilities (>= v3.3.19.0)

6.6 IotMqttSampleAzureIotHub

Sample of MQTT communication with the Microsoft Azure IoT Hub

This sample shows 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

The topic structure for sending and receiving messages is predefined by the Microsoft Azure IoT Hub.

Authentication

You can use either a SAS token or X509 certificates to authenticate the MQTT client.

QoS and Retain

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 IotMqttSampleUsingQueue [▶ 312] sample, 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;

  (* Option 1: authentication via Device SAS Token *)
  fbMqttClient.stTLS.sCA := 'c:
\TwinCAT\3.1\Config\Certificates\DigiCertGlobalRootG2.cer'; // CA certificate
  fbMqttClient.stTLS.sAzureSas := 'PlaceDeviceSasTokenHere'; // Device SAS Token

  (* Option 2: authentication via X509 certificate *)
  //fbMqttClient.stTLS.sCA := 'c:
\TwinCAT\3.1\Config\Certificates\DigiCertGlobalRootG2.cer'; // CA certificate
  //fbMqttClient.stTLS.sCert := 'c:\TwinCAT\3.1\Config\Certificates\MyDeviceCert.pem';
  //fbMqttClient.stTLS.sKeyFile := 'c:\TwinCAT\3.1\Config\Certificates\MyDeviceCert.key';
  //fbMqttClient.sHostName:= myIotHub.azure-devices.net';
  //fbMqttClient.nHostPort:= 8883;
  //fbMqttClient.sClientId := 'MyDevice';
  //fbMqttClient.sUserName := 'myIotHub.azure-devices.net/MyDevice/?api-version=2021-04-12';
   fbMqttClient.ipMessageQueue := fbMessageQueue;
END_IF
```

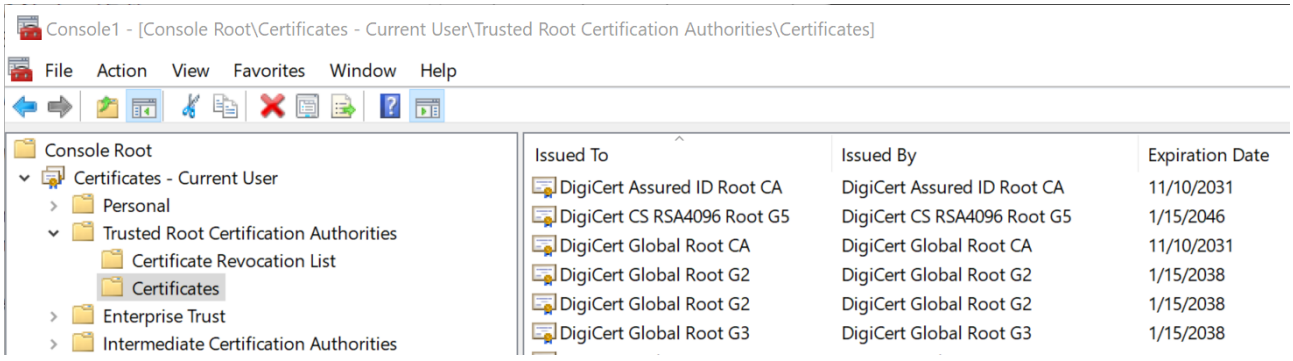
You can use either a SAS token or X509 certificates to authenticate the MQTT client on the Azure IoT hub. Depending on the authentication type, certain connection parameters must be set. The following table provides an overview of the parameters to be set.

	SAS token	X509 certificate
stTLS.sCA	Path to the CA certificate used by the Azure IoT Hub. See below.	Path to the CA certificate used by the Azure IoT Hub. See below.
stTLS.sAzureSas	SAS token, which can be generated via the Azure File Explorer.	---
stTLS.sCert	---	Path to the client certificate.
stTLS.sKey	---	Path to the private key.
sHostname	Does not have to be set explicitly. The host name is derived from the SAS token.	Host name of the Azure IoT Hub instance.
nHostPort	Does not have to be set explicitly. The port is automatically set to 8883 when using a SAS token.	8883
sClientId	Does not have to be set explicitly. The client ID is derived from the SAS token.	Corresponds to the device name of the created device in the Azure IoT Hub.
sUserName	Does not have to be set explicitly. The user name is derived from the SAS token.	Corresponds to a fixed naming scheme, which includes both the IoT hub name and device name. For more information, please refer to the Azure IoT Hub documentation.

CA certificate

When establishing a connection to the Microsoft Azure IoT Hub via MQTT, the specification of a CA certificate is mandatory, both when using a SAS token and when using X509 certificates for client authentication. Please always consult the Microsoft documentation to find out the currently valid root CA.

In most cases, the associated public key of the respective root CA can be extracted from the Microsoft Windows Certificate Console (**Start** > **Run** > **mmc.exe**, then add the **SnapIn** "Certificates"). The root CA can then be found under the heading **Trusted Root Certification Authorities.** See also screenshot below for comparison.

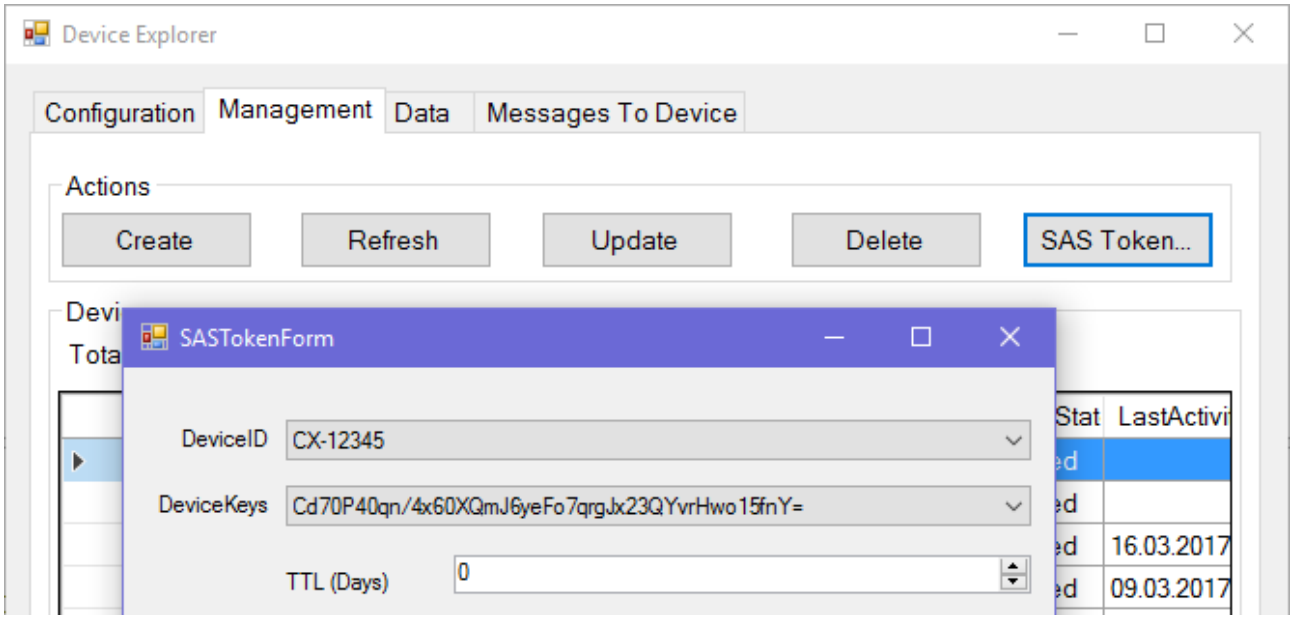


Publish

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

When 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_IotBase, Tc2_Utilities (>= v3.3.19.0)

6.7 IotMqttSampleBoschIoT

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 IotMqttSampleUsingQueue [▶ 312], 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 IotMqttSampleIbmWatsonIoT

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 IotMqttSampleUsingQueue [▶ 312], 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_Utilities (>= v3.3.19.0)

6.9 IotMqttSampleMathworksThingspeak

Communication with the MathWorks ThingSpeak Cloud is illustrated in this sample. Since this sample is essentially based on the sample IotMqttSampleUsingQueue [▶ 312], 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.sUserPassword:= '<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.sUserPassword:= '<password as provided by MQTT Device on ThingSpeak>';

  stMqttTls.sVersion:= 'tlsv1.2';
  stMqttTls.sCA:= 'C:\TwinCAT\3.1\Config\Certificates\<thingSpeakCert.pem>;
  fbMqttClient.stTLS := stMqttTls;
END_IF
```

Publish

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_IotBase, Tc2_Utilities (>= v3.3.19.0)

6.10 IotMqttSampleAzureIotHubDeviceTwin

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 IotMqttSampleAzureIotHub [▶ 320]. 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

6.11 IotMqttv5Sample

Sample for MQTTv5 communication via a message queue

In this sample, the communication to an MQTT broker using MQTTv5 is shown. As a prerequisite, the message broker used must support MQTTv5. Messages are sent ("Publish") and received ("Subscribe"). This is done in two steps. First, the topic is used to decide which messages are to be received. Then, the received messages are collected in a message queue, from where they can be read and evaluated. The sample sends and receives messages on the same topic for demonstration purposes.

Handling the MessageQueue

In contrast to the FB_IotMqttClient function block for MQTTv3, in the FB_IotMqtt5Client function block you no longer have to instantiate the message queue separately as an object and link it to the function block. Instead, you can directly access the corresponding output of the function block to work with the message queue.

Project structure

1. Create a TwinCAT project with a PLC and attach the Tc3_IotBase and Tc3_JsonXml as library references.
2. Create a program block and declare an instance of FB_IotMqtt5ClientBase [▶ 79] and two auxiliary variables to control the program sequence, if required.

```
PROGRAM PrgMqttCom
VAR
    fbMqttClient    : FB_IotMqtt5Client;
    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. Three imaginary sensor values are used for the message, which are to be packaged into a JSON document via the F_CreateMessage function, which is yet to be created.

```
    (* published message *)
    sTopicPub : STRING(255) := 'MyTopic';
    sPayloadPub : STRING(255);
    fbTimer : TON := (PT:=T#1S);
    rSensor1 : REAL;
    nSensor2 : DINT;
    bSensor3 : BOOL;
```

4. For message reception, declare a variable containing the topic to which the client should subscribe and two other variables indicating the topic and payload of the last received message.
The received messages should be collected in a queue to be evaluated one after the other. For this you declare an instance of FB_IotMqtt5MessageQueue and an instance of FB_IotMqtt5Message.

```
    (* received message *)
    bSubscribed     : BOOL;
    sTopicSub       : STRING(255) := 'MyTopic';
    {attribute 'TcEncoding':='UTF-8'}
    sTopicRcv       : STRING(255);
    {attribute 'TcEncoding':='UTF-8'}
    sPayloadRcv     : STRING(255);
    fbMessage       : FB_IotMqtt5Message;
```

5. 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;
END_IF

fbMqttClient.Execute(bConnect);
```

6. As soon as the connection to the broker is established, the client should subscribe to a particular topic. Likewise, a message is to be sent every second.
In the sample `sTopicPub = sTopicSub`, so that a loopback is created. In other applications the topics usually differ.

```
IF fbMqttClient.bConnected THEN
    IF NOT bSubscribed THEN
        bSubscribed := fbMqttClient.Subscribe(sTopic:=sTopicSub, eQoS:=TcIotMqttQos.AtMostOnceDe
livery);
    END_IF
    fbTimer(IN:=TRUE);
    IF fbTimer.Q THEN // publish new payload every second
        fbTimer(IN:=FALSE);
        rSensor1 := rSensor1 + 0.1;
        nSensor2 := nSensor2 + 1;
        bSensor3 := NOT bSensor3;
        sPayloadPub := F_CreateMessage(rSensor1, nSensor2, bSensor3);
        fbMqttClient.Publish(sTopic:= sTopicPub,
pPayload:= ADR(sPayloadPub), nPayloadSize:= LEN2(ADR(sPayloadPub))
+1, eQoS:= TcIotMqttQos.AtMostOnceDelivery, bRetain:= FALSE, bQueue:= FALSE );
    END_IF
END_IF
```

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

8. 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.
9. Finally, add a new function called F_CreateMessage. The function shall have the return value STRING(255) and shall be provided with the following function signature:

```
FUNCTION F_CreateMessage : STRING(255)
VAR_INPUT
  Sensor1 : REAL;
  Sensor2 : DINT;
  Sensor3 : BOOL;
END_VAR
VAR
  dtTimestamp : DATE_AND_TIME;
  timeAsFileTime : T_FILETIME64;
  fbJson : FB_JsonSaxWriter;
END_VAR
```

10. The function first generates a timestamp based on the F_GetSystemTime function and then packages the passed sensor values and timestamp into a JSON document.

```
timeAsFileTime := F_GetSystemTime();
dtTimestamp := FILETIME64_TO_DT( timeAsFileTime );
fbJson.StartObject();
fbJson.AddKey('Timestamp');
fbJson.AddDateTime(dtTimestamp);
fbJson.AddKey('Values');
fbJson.StartObject();
fbJson.AddKey('Sensor1');
fbJson.AddReal(Sensor1);
fbJson.AddKey('Sensor2');
fbJson.AddDint(Sensor2);
fbJson.AddKey('Sensor3');
fbJson.AddBool(Sensor3);
fbJson.EndObject();
fbJson.EndObject();
F_CreateMessage := fbJson.GetDocument();
fbJson.ResetDocument();
```

Further steps

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 fbMqttClient.fbMessageQueue.nQueuedMessages > 0 THEN
    IF fbMqttClient.fbMessageQueue.Dequeue(fbMessage:=fbMessage) THEN
        IF fbMessage.CompareTopic(sTopic:=sTopicSub1) THEN
            fbMessage.GetPayload(pPayload:=ADR(sPayloadRcv1), nPayloadSize:=SIZEOF(sPayloadRcv1), bS
etNullTermination:=FALSE);
        ELSIF fbMessage.CompareTopic(sTopic:=sTopicSub2) THEN
            fbMessage.GetPayload(pPayload:=ADR(sPayloadRcv2), nPayloadSize:=SIZEOF(sPayloadRcv2), bS
etNullTermination:=FALSE);
        END_IF
    END_IF
END_IF
```

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4026.0	IPC or CX (x86, x64, ARM)	Tc3_IotBase (>= v3.4.2.0), Tc2_Utilities (>= v3.4.4.0)

6.12 IotMqttv5LastWillSample

Sample for the use of MQTTv5 UserProperties

The communication to an MQTT broker using MQTTv5 is shown in this sample. In particular, the LastWill mechanism of MQTTv5 is demonstrated. As a prerequisite, the message broker used must support MQTTv5. The basic flow of the sample is as follows:

- There is a PLC project that connects to a local message broker based on MQTTv5 and publishes messages to it. At the same time, the application subscribes to the same topic to receive the sent messages again. The structure of the program corresponds to the sample IotMqttv5Sample [▶ 325].
- When a connection is established, a LastWill message is specified and some properties are set for the LastWill message. The LastWill is then transmitted to the message broker in the event of a Connect, and to interested clients in the event of a Disconnect.

The following code snippet once again shows the relevant part for specifying the LastWill message, as well as the associated properties. This code position is used for the initialization (only once) of the connection parameters.

```
fbMqttClient.stWill.sTopic := 'MyLastWillTopic';
fbMqttClient.stWill.sContentType := 'MyContentType';
fbMqttClient.stWill.eQoS := TcIotMqttQoS.ExactlyOnceDelivery;
fbMqttClient.stWill.fbPayload.SetData(ADR(sLastWillMsg), SIZEOF(sLastWillMsg));
fbMqttClient.stWill.fbUserProperties.AddUserProperty('MyFirst', 'UserProperty');
```

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4026.0	IPC or CX (x86, x64, ARM)	Tc3_IotBase (>= v3.4.2.0), Tc2_Utilities (>= v3.4.4.0)

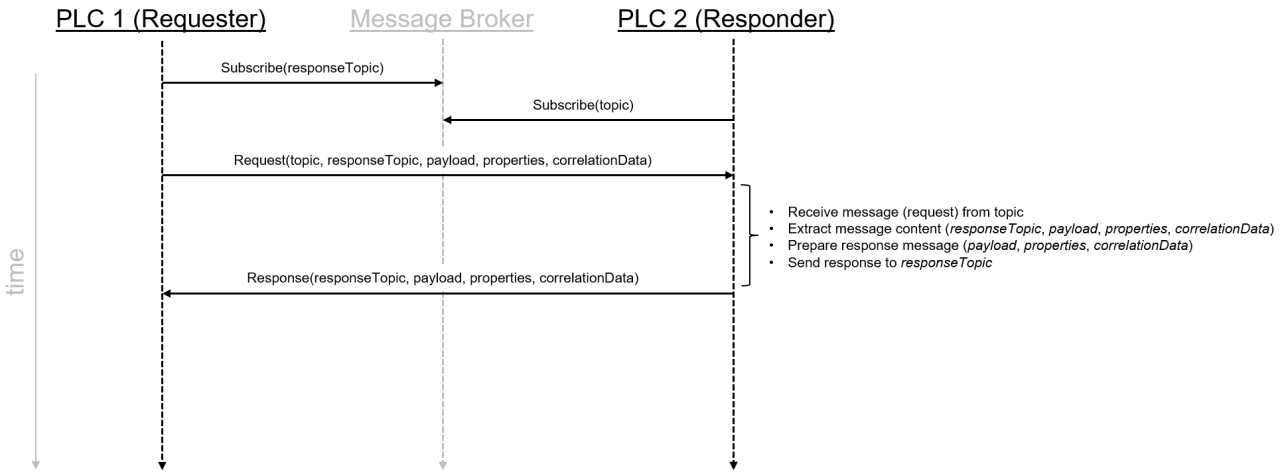
6.13 IotMqttv5ReqResSample

Sample for the use of MQTTv5 Request/Response

In this sample, the communication to an MQTT broker using MQTTv5 is shown. In particular, the Request/Response [▶ 33] mechanism of MQTTv5 is demonstrated. As a prerequisite, the message broker used must support MQTTv5. The basic flow of the sample is as follows:

- There is a PLC project that demonstrates the request function and also receives and evaluates the response from the remote terminal. The project first subscribes to the so-called response topic (on which it expects a response from the remote terminal) and then sends the request, which contains the name of the response topic. Advanced properties of the MQTTv5 message, such as UserProperties or CorrelationData, are also demonstrated here.
- A second PLC project is responsible for receiving the request and sending a corresponding response to the response topic. The MQTT client first subscribes to the topic on which the request is received and prepares the corresponding response when a request is received. The response topic and the extended properties of the MQTTv5 message are extracted from the received message and used for the response.

The following figure once again illustrates the basic flow of communication. More information about individual lines of code can be found directly in the comments of the corresponding sample download.



Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4026.0	IPC or CX (x86, x64, ARM)	Tc3_IotBase (>= v3.4.2.0), Tc2_Utilities (>= v3.4.4.0)

6.14 IotMqttv5UserPropsSample

Sample for the use of MQTTv5 UserProperties

The communication to an MQTT broker using MQTTv5 is shown in this sample. In particular, the UserProperties mechanism of MQTTv5 is demonstrated. As a prerequisite, the message broker used must support MQTTv5. The basic flow of the sample is as follows:

- There is a PLC project that connects to a local message broker based on MQTTv5 and publishes messages to it. At the same time, the application subscribes to the same topic to receive the sent messages again.
- The structure of the program corresponds to the sample IotMqttv5Sample [▶ 325]. In addition to the actual payload, the messages also contain UserProperties, the handling of which is demonstrated both when sending and receiving messages.

The following code snippet once again shows the relevant part for specifying the User Properties on a message to be sent.

```
fbPubProps.sContentType := sContentType;
fbPubProps.nMsgExpiryInterval := 7;
fbPubProps.bPayloadUtf8 := TRUE;
fbPubProps.ClearUserProperties();
FOR m:=1 TO 10 DO
  hrPropSet := fbPubProps.AddUserProperty(aUserName[m], aUserValue[m]);
  IF FAILED(hrPropSet) THEN
    EXIT;
  END_IF
END_FOR
```

In this sample, 10 User Properties whose key/value values are taken from the aUserName and aUserValue arrays are thus added to a message to be sent. The user properties are then passed to the message to be sent as input parameters to the Publish() method.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4026.0	IPC or CX (x86, x64, ARM)	Tc3_IotBase (>= v3.4.2.0), Tc2_Utilities (>= v3.4.4.0)

6.15 JsonXmlSamples

6.15.1 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.

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
```

6.15.2 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.

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);
```

6.15.3 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.

Declaration range

```
PROGRAM MAIN
VAR
  fbJson     : FB_JsonDomParser;
  jsonDoc    : SJsonValue;
  jsonProp   : SJsonValue;
  jsonValue  : SJsonValue;
  bHasMember : BOOL;
  sMessage   : STRING(255) := '{"serialNumber":"G030PT028191AC4R","batteryVoltage":"1547mV","clickTy
pe":"SINGLE"}';
  stReceivedData : ST_ReceivedData;
END_VAR
```

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
```

6.15.4 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.

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 though 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
  }
}
```

6.15.5 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.

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.

6.15.6 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.

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(stValu
es));
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 Sample configurations

7.1.1 Mosquitto

Below are some sample configurations for the Mosquitto Message Broker. The configurations were created based on the official Mosquitto documentation. The configuration file (mosquitto.conf) of the broker can be found in the installation directory of the broker (for example C:\Program Files\mosquitto). This can be opened and edited with a text editor of your choice, for example Notepad.

We recommend creating a backup before making changes to the configuration file. Alternatively, you can create additional configuration files and load them from the command line when starting the broker:

```
mosquitto.exe -c mosquitto.conf
```

For debugging and testing purposes, we recommend starting the broker in a command line, since verbose output can also be enabled here:

```
mosquitto.exe -c mosquitto.conf -v
```

Mosquitto start in the command line

Before starting Mosquitto Broker from the command line, please make sure that no active Mosquitto process is running on your system. This may be the case, for example, if Mosquitto Broker was installed and started as a Windows service. Therefore, please check Windows Task Manager and Windows Services Manager.

In the following you will now find some sample configurations for different use cases. These configurations were created based on Mosquitto Message Broker version 2.0.15 and are also available in our TF6701_Samples repository on GitHub.

Unsecured connection

The following configuration file content configures the broker for an unsecured connection without TLS and without user authentication.

```
listener 1883
allow_anonymous true
```

Unsecured connection with user authentication

The following configuration file content configures the broker for an unsecured connection without TLS, but with user authentication.

```
listener 1883
allow_anonymous false
password_file C:\Program Files\mosquitto\users.pwd
```

The user database specified with the password_file parameter can be created with the mosquitto_passwd tool, which is also located in the broker installation directory. The following call creates a new user database and adds the user MyUser1 with the password SecurePassword:

```
mosquitto_passwd.exe -b -c "C:\Program Files\mosquitto\users.pwd" MyUser1 SecurePassword
```

By using the -b parameter, a password can be passed directly in the call. The -c parameter creates a new password file. If a new user is to be added to an existing file, this parameter is omitted.

```
mosquitto_passwd.exe -b "C:\Program Files\mosquitto\users.pwd" MyUser2 AnotherSecurePassword
```

Plain text transmission of the password

Please note that in case of an insecure, i.e. non-encrypted MQTT connection, the password is transmitted in plain text during user authentication. We therefore recommend encrypting the MQTT communication using TLS, see sample configurations below.

Unsecured connection with user authentication and ACL

The following configuration file content configures the broker for an unsecured connection without TLS, but with user authentication and using an Access Control List (ACL) that regulates user access to specific topics.

```
listener 1883
allow_anonymous false
password_file C:\Program Files\mosquitto\users.pwd
acl_file C:\Program Files\mosquitto\userAccess.acl
```

As described in the previous sample, the user database referenced as password_file can be created using the mosquitto_passwd tool. The access list file referenced as acl_file is a plain text file which is constructed according to a specific format. This format is also described in more detail in the Mosquitto documentation. The following file shows a sample of what an access list for two users might look like. Each user is granted access to his own topic area. However, an administrator user should have access to all topics.

```
user Admin
topic #
user MyUser1
topic users/MyUser1/#
user MyUser2
topic users/MyUser2/#
```

Access to a topic that is not allowed

Please note that the client does not receive any feedback from the Mosquitto Message Broker when accessing (Publish/Subscribe) a topic that is not allowed. From the client's point of view, the process appears to be successful, but the broker does not transmit or forward any messages to the client other than the regular MQTT command packages.

TLS connection with PSK

The following configuration file content configures the broker for a secured connection with TLS using a pre-shared key (PSK). No user authentication is configured.

```
listener 8883
allow_anonymous true
psk_hint TwinCATrocks
psk_file C:\Program Files\mosquitto\myKeys.psk
```

The file referenced as psk_file then contains a list of pre-shared keys. This is a plain text file in which the PSKs are listed line by line in the identity:key format. The key is specified in hexadecimal format. Example:

```
MyIdentity1:ab123456789cd
MyIdentity2:ef987654321ab
```

The psk_hint parameter is important because it enables TLS-PSK for the listener.

TLS connection with certificate

The following configuration file content configures the broker for a secured connection with TLS using a certificate. No user authentication is configured.

This sample assumes that you have a Certificate Authority (CA) that can issue server certificates. There are various tutorials on the web which demonstrate, for example, how you can create such a CA using OpenSSL and issue corresponding certificates.

```
listener 8883
allow_anonymous true
cafile C:\ca\certs\fullChain.pem
crlfile C:\ca\crl\intermediateCA.crl
certfile C:\Program Files\mosquitto\certs\mosquitto.pem
keyfile C:\Program Files\mosquitto\certs\mosquitto.key
```

TLS connection with certificate and user authentication

The following configuration file content configures the broker for a secured connection with TLS using a certificate. In addition, user authentication is configured specifying the user database file.

This sample assumes that you have a Certificate Authority (CA) that can issue server certificates. There are various tutorials on the web which demonstrate, for example, how you can create such a CA using OpenSSL and issue corresponding certificates.

```
listener 8883
allow_anonymous false
password_file C:\Program Files\mosquitto\users.pwd
cafile C:\ca\certs\fullChain.pem
crlfile C:\ca\crl\intermediateCA.crl
certfile C:\Program Files\mosquitto\certs\mosquitto.pem
keyfile C:\Program Files\mosquitto\certs\mosquitto.key
```

Clients attempting to connect to this broker must have a valid certificate issued by the same Certificate Authority.

TLS connection with certificate, user authentication and ACL

The following configuration file content configures the broker for a secured connection with TLS using a certificate. In addition, user authentication is configured specifying the user database file, as well as an Access Control List (ACL) which configures various access rights for the users (see sample above).

This sample assumes that you have a Certificate Authority (CA) that can issue server certificates. There are various tutorials on the web which demonstrate, for example, how you can create such a CA using OpenSSL and issue corresponding certificates.

```
listener 8883
allow_anonymous false
use_identity_as_username true
password_file C:\Program Files\mosquitto\users.pwd
acl_file C:\Program Files\mosquitto\userAccess.acl
cafile C:\ca\certs\fullChain.pem
crlfile C:\ca\crl\intermediateCA.crl
certfile C:\Program Files\mosquitto\certs\mosquitto.pem
keyfile C:\Program Files\mosquitto\certs\mosquitto.key
```

Clients attempting to connect to this broker must have a valid certificate issued by the same Certificate Authority.

Building on this sample, you can now also use the use_identity_as_username parameter, which causes the CommonName from the client certificate to be used as the user name.

7.2 Error Codes

In the event of an error, the function block FB_IotMqttClient [▶ 55] sets the output bError and indicates the error with hrErrorCode. All errors are listed in section "ADS Return Codes [▶ 340]".

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
```

7.3 ADS Return Codes

Grouping of error codes:
Global error codes: ADS Return Codes [▶ 340]... (0x9811_0000 ...)
Router error codes: ADS Return Codes [▶ 340]... (0x9811_0500 ...)
General ADS errors: ADS Return Codes [▶ 341]... (0x9811_0700 ...)
RTime error codes: ADS Return Codes [▶ 343]... (0x9811_1000 ...)

Global error codes

Hex	Dec	HRESULT	Name	Description
0x0	0	0x98110000	ERR_NOERROR	No error.
0x1	1	0x98110001	ERR_INTERNAL	Internal error.
0x2	2	0x98110002	ERR_NORTIME	No real time.
0x3	3	0x98110003	ERR_ALLOCLOCKEDMEM	Allocation locked – memory error.
0x4	4	0x98110004	ERR_INSERTMAILBOX	Mailbox full – the ADS message could not be sent. Reducing the number of ADS messages per cycle will help.
0x5	5	0x98110005	ERR_WRONGRECEIVEHMSG	Wrong HMSG.
0x6	6	0x98110006	ERR_TARGETPORTNOTFOUND	Target port not found – ADS server is not started or is not reachable.
0x7	7	0x98110007	ERR_TARGETMACHINENOTFOUND	Target computer not found – AMS route was not found.
0x8	8	0x98110008	ERR_UNKNOWNCMDID	Unknown command ID.
0x9	9	0x98110009	ERR_BADTASKID	Invalid task ID.
0xA	10	0x9811000A	ERR_NOIO	No IO.
0xB	11	0x9811000B	ERR_UNKNOWNAMSCMD	Unknown AMS command.
0xC	12	0x9811000C	ERR_WIN32ERROR	Win32 error.
0xD	13	0x9811000D	ERR_PORTNOTCONNECTED	Port not connected.
0xE	14	0x9811000E	ERR_INVALIDAMSLENGTH	Invalid AMS length.
0xF	15	0x9811000F	ERR_INVALIDAMSNETID	Invalid AMS Net ID.
0x10	16	0x98110010	ERR_LOWINSTLEVEL	Installation level is too low –TwinCAT 2 license error.
0x11	17	0x98110011	ERR_NODEBUGINTAVAILABLE	No debugging available.
0x12	18	0x98110012	ERR_PORTDISABLED	Port disabled – TwinCAT system service not started.
0x13	19	0x98110013	ERR_PORTALREADYCONNECTED	Port already connected.
0x14	20	0x98110014	ERR_AMSSYNC_W32ERROR	AMS Sync Win32 error.
0x15	21	0x98110015	ERR_AMSSYNC_TIMEOUT	AMS Sync Timeout.
0x16	22	0x98110016	ERR_AMSSYNC_AMSERROR	AMS Sync error.
0x17	23	0x98110017	ERR_AMSSYNC_NOINDEXINMAP	No index map for AMS Sync available.
0x18	24	0x98110018	ERR_INVALIDAMSPORT	Invalid AMS port.
0x19	25	0x98110019	ERR_NOMEMORY	No memory.
0x1A	26	0x9811001A	ERR_TCPSEND	TCP send error.
0x1B	27	0x9811001B	ERR_HOSTUNREACHABLE	Host unreachable.
0x1C	28	0x9811001C	ERR_INVALIDAMSFRAGMENT	Invalid AMS fragment.
0x1D	29	0x9811001D	ERR_TLSSEND	TLS send error – secure ADS connection failed.
0x1E	30	0x9811001E	ERR_ACCESSDENIED	Access denied – secure ADS access denied.

Router error codes

Hex	Dec	HRESULT	Name	Description
0x500	1280	0x98110500	ROUTERERR_NOLOCKEDMEMORY	Locked memory cannot be allocated.
0x501	1281	0x98110501	ROUTERERR_RESIZEMEMORY	The router memory size could not be changed.
0x502	1282	0x98110502	ROUTERERR_MAILBOXFULL	The mailbox has reached the maximum number of possible messages.
0x503	1283	0x98110503	ROUTERERR_DEBUGBOXFULL	The Debug mailbox has reached the maximum number of possible messages.
0x504	1284	0x98110504	ROUTERERR_UNKNOWNPORTTYPE	The port type is unknown.
0x505	1285	0x98110505	ROUTERERR_NOTINITIALIZED	The router is not initialized.
0x506	1286	0x98110506	ROUTERERR_PORTALREADYINUSE	The port number is already assigned.
0x507	1287	0x98110507	ROUTERERR_NOTREGISTERED	The port is not registered.
0x508	1288	0x98110508	ROUTERERR_NOMOREQUEUES	The maximum number of ports has been reached.
0x509	1289	0x98110509	ROUTERERR_INVALIDPORT	The port is invalid.
0x50A	1290	0x9811050A	ROUTERERR_NOTACTIVATED	The router is not active.
0x50B	1291	0x9811050B	ROUTERERR_FRAGMENTBOXFULL	The mailbox has reached the maximum number for fragmented messages.
0x50C	1292	0x9811050C	ROUTERERR_FRAGMENTTIMEOUT	A fragment timeout has occurred.
0x50D	1293	0x9811050D	ROUTERERR_TOBEREMOVED	The port is removed.

General ADS error codes

Hex	Dec	HRESULT	Name	Description
0x700	1792	0x98110700	ADSERR_DEVICE_ERROR	General device error.
0x701	1793	0x98110701	ADSERR_DEVICE_SRVNOTSUPP	Service is not supported by the server.
0x702	1794	0x98110702	ADSERR_DEVICE_INVALIDGRP	Invalid index group.
0x703	1795	0x98110703	ADSERR_DEVICE_INVALIDOFFSET	Invalid index offset.
0x704	1796	0x98110704	ADSERR_DEVICE_INVALIDACCESS	Reading or writing not permitted.
0x705	1797	0x98110705	ADSERR_DEVICE_INVALIDSIZE	Parameter size not correct.
0x706	1798	0x98110706	ADSERR_DEVICE_INVALIDDATA	Invalid data values.
0x707	1799	0x98110707	ADSERR_DEVICE_NOTREADY	Device is not ready to operate.
0x708	1800	0x98110708	ADSERR_DEVICE_BUSY	Device is busy.
0x709	1801	0x98110709	ADSERR_DEVICE_INVALIDCONTEXT	Invalid operating system context. This can result from use of ADS blocks in different tasks. It may be possible to resolve this through multitasking synchronization in the PLC.
0x70A	1802	0x9811070A	ADSERR_DEVICE_NOMEMORY	Insufficient memory.
0x70B	1803	0x9811070B	ADSERR_DEVICE_INVALIDPARM	Invalid parameter values.
0x70C	1804	0x9811070C	ADSERR_DEVICE_NOTFOUND	Not found (files, ...).
0x70D	1805	0x9811070D	ADSERR_DEVICE_SYNTAX	Syntax error in file or command.
0x70E	1806	0x9811070E	ADSERR_DEVICE_INCOMPATIBLE	Objects do not match.
0x70F	1807	0x9811070F	ADSERR_DEVICE_EXISTS	Object already exists.
0x710	1808	0x98110710	ADSERR_DEVICE_SYMBOLNOTFOUND	Symbol not found.
0x711	1809	0x98110711	ADSERR_DEVICE_SYMBOLVERSIONINVALID	Invalid symbol version. This can occur due to an online change. Create a new handle.
0x712	1810	0x98110712	ADSERR_DEVICE_INVALIDSTATE	Device (server) is in invalid state.
0x713	1811	0x98110713	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransMode not supported.
0x714	1812	0x98110714	ADSERR_DEVICE_NOTIFYHNDINVALID	Notification handle is invalid.
0x715	1813	0x98110715	ADSERR_DEVICE_CLIENTUNKNOWN	Notification client not registered.
0x716	1814	0x98110716	ADSERR_DEVICE_NOMOREHDLS	No further handle available.
0x717	1815	0x98110717	ADSERR_DEVICE_INVALIDWATCHSIZE	Notification size too large.
0x718	1816	0x98110718	ADSERR_DEVICE_NOTINIT	Device not initialized.
0x719	1817	0x98110719	ADSERR_DEVICE_TIMEOUT	Device has a timeout.
0x71A	1818	0x9811071A	ADSERR_DEVICE_NOINTERFACE	Interface query failed.
0x71B	1819	0x9811071B	ADSERR_DEVICE_INVALIDINTERFACE	Wrong interface requested.
0x71C	1820	0x9811071C	ADSERR_DEVICE_INVALIDCLSID	Class ID is invalid.
0x71D	1821	0x9811071D	ADSERR_DEVICE_INVALIDOBJID	Object ID is invalid.
0x71E	1822	0x9811071E	ADSERR_DEVICE_PENDING	Request pending.
0x71F	1823	0x9811071F	ADSERR_DEVICE_ABORTED	Request is aborted.
0x720	1824	0x98110720	ADSERR_DEVICE_WARNING	Signal warning.
0x721	1825	0x98110721	ADSERR_DEVICE_INVALIDARRAYIDX	Invalid array index.
0x722	1826	0x98110722	ADSERR_DEVICE_SYMBOLNOTACTIVE	Symbol not active.
0x723	1827	0x98110723	ADSERR_DEVICE_ACCESSDENIED	Access denied.
0x724	1828	0x98110724	ADSERR_DEVICE_LICENSENOTFOUND	Missing license.
0x725	1829	0x98110725	ADSERR_DEVICE_LICENSEEXPIRED	License expired.
0x726	1830	0x98110726	ADSERR_DEVICE_LICENSEEXCEEDED	License exceeded.
0x727	1831	0x98110727	ADSERR_DEVICE_LICENSEINVALID	Invalid license.
0x728	1832	0x98110728	ADSERR_DEVICE_LICENSESYSTEMID	License problem: System ID is invalid.
0x729	1833	0x98110729	ADSERR_DEVICE_LICENSENOTIMELIMIT	License not limited in time.
0x72A	1834	0x9811072A	ADSERR_DEVICE_LICENSEFUTUREISSUE	Licensing problem: time in the future.
0x72B	1835	0x9811072B	ADSERR_DEVICE_LICENSETIMETOLONG	License period too long.
0x72C	1836	0x9811072C	ADSERR_DEVICE_EXCEPTION	Exception at system startup.
0x72D	1837	0x9811072D	ADSERR_DEVICE_LICENSEDUPLICATED	License file read twice.
0x72E	1838	0x9811072E	ADSERR_DEVICE_SIGNATUREINVALID	Invalid signature.
0x72F	1839	0x9811072F	ADSERR_DEVICE_CERTIFICATEINVALID	Invalid certificate.
0x730	1840	0x98110730	ADSERR_DEVICE_LICENSEOEMNOTFOUND	Public key not known from OEM.
0x731	1841	0x98110731	ADSERR_DEVICE_LICENSERESTRICTED	License not valid for this system ID.
0x732	1842	0x98110732	ADSERR_DEVICE_LICENSEDEMODENIED	Demo license prohibited.
0x733	1843	0x98110733	ADSERR_DEVICE_INVALIDFNCID	Invalid function ID.
0x734	1844	0x98110734	ADSERR_DEVICE_OUTOFRANGE	Outside the valid range.
0x735	1845	0x98110735	ADSERR_DEVICE_INVALIDALIGNMENT	Invalid alignment.
0x736	1846	0x98110736	ADSERR_DEVICE_LICENSEPLATFORM	Invalid platform level.

Hex	Dec	HRESULT	Name	Description
0x737	1847	0x98110737	ADSERR_DEVICE_FORWARD_PL	Context – forward to passive level.
0x738	1848	0x98110738	ADSERR_DEVICE_FORWARD_DL	Context – forward to dispatch level.
0x739	1849	0x98110739	ADSERR_DEVICE_FORWARD_RT	Context – forward to real time.
0x740	1856	0x98110740	ADSERR_CLIENT_ERROR	Client error.
0x741	1857	0x98110741	ADSERR_CLIENT_INVALIDPARM	Service contains an invalid parameter.
0x742	1858	0x98110742	ADSERR_CLIENT_LISTEMPTY	Polling list is empty.
0x743	1859	0x98110743	ADSERR_CLIENT_VARUSED	Var connection already in use.
0x744	1860	0x98110744	ADSERR_CLIENT_DUPLINVOKEID	The called ID is already in use.
0x745	1861	0x98110745	ADSERR_CLIENT_SYNCTIMEOUT	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	0x98110746	ADSERR_CLIENT_W32ERROR	Error in Win32 subsystem.
0x747	1863	0x98110747	ADSERR_CLIENT_TIMEOUTINVALID	Invalid client timeout value.
0x748	1864	0x98110748	ADSERR_CLIENT_PORTNOTOPEN	Port not open.
0x749	1865	0x98110749	ADSERR_CLIENT_NOAMSADDR	No AMS address.
0x750	1872	0x98110750	ADSERR_CLIENT_SYNCINTERNAL	Internal error in Ads sync.
0x751	1873	0x98110751	ADSERR_CLIENT_ADDHASH	Hash table overflow.
0x752	1874	0x98110752	ADSERR_CLIENT_REMOVEHASH	Key not found in the table.
0x753	1875	0x98110753	ADSERR_CLIENT_NOMORESYM	No symbols in the cache.
0x754	1876	0x98110754	ADSERR_CLIENT_SYNCRESINVALID	Invalid response received.
0x755	1877	0x98110755	ADSERR_CLIENT_SYNCPORTLOCKED	Sync Port is locked.
0x756	1878	0x98110756	ADSERR_CLIENT_REQUESTCANCELLED	The request was cancelled.

RTime error codes

Hex	Dec	HRESULT	Name	Description
0x1000	4096	0x98111000	RTERR_INTERNAL	Internal error in the real-time system.
0x1001	4097	0x98111001	RTERR_BADTIMERPERIODS	Timer value is not valid.
0x1002	4098	0x98111002	RTERR_INVALIDTASKPTR	Task pointer has the invalid value 0 (zero).
0x1003	4099	0x98111003	RTERR_INVALIDSTACKPTR	Stack pointer has the invalid value 0 (zero).
0x1004	4100	0x98111004	RTERR_PRIOEXISTS	The request task priority is already assigned.
0x1005	4101	0x98111005	RTERR_NOMORETCB	No free TCB (Task Control Block) available. The maximum number of TCBs is 64.
0x1006	4102	0x98111006	RTERR_NOMORESEMAS	No free semaphores available. The maximum number of semaphores is 64.
0x1007	4103	0x98111007	RTERR_NOMOREQUEUES	No free space available in the queue. The maximum number of positions in the queue is 64.
0x100D	4109	0x9811100D	RTERR_EXTIRQALREADYDEF	An external synchronization interrupt is already applied.
0x100E	4110	0x9811100E	RTERR_EXTIRQNOTDEF	No external sync interrupt applied.
0x100F	4111	0x9811100F	RTERR_EXTIRQINSTALLFAILED	Application of the external synchronization interrupt has failed.
0x1010	4112	0x98111010	RTERR_IRQLNOTLESSOREQUAL	Call of a service function in the wrong context
0x1017	4119	0x98111017	RTERR_VMXNOTSUPPORTED	Intel VT-x extension is not supported.
0x1018	4120	0x98111018	RTERR_VMXDISABLED	Intel VT-x extension is not enabled in the BIOS.
0x1019	4121	0x98111019	RTERR_VMXCONTROLSMISSING	Missing function in Intel VT-x extension.
0x101A	4122	0x9811101A	RTERR_VMXENABLEFAILS	Activation of Intel VT-x fails.

Specific positive HRESULT Return Codes:

HRESULT	Name	Description
0x0000_0000	S_OK	No error.
0x0000_0001	S_FALSE	No error. Example: successful processing, but with a negative or incomplete result.
0x0000_0203	S_PENDING	No error. Example: successful processing, but no result is available yet.
0x0000_0256	S_WATCHDOG_TIMEOUT	No error. Example: successful processing, but a timeout occurred.

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	WSAEHOSTUNREACH	No route to host - a socket operation referred to an unavailable host.
More Winsock error codes: Win32 error codes			

7.4 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	Check whether the message broker can be reached in principle from the system on which the Tc3_IotBase is executed (e.g. using another MQTT client). 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_IotBase). On the message broker side, these ports must be open for incoming messages. If the Tc3_IotBase is used, the port configuration can be checked via the input parameter nHostPort at the function block FB_IotMqttClient [▶ 55].
Connection to AWS IoT cannot be established	Establishing the connection	Check whether the AWS IoT URL can be reached in principle from the system on which the Tc3_IotBase 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. 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_IotBase). If the Tc3_IotBase is used, the port configuration can be checked via the input parameter nHostPort at the function block FB_IotMqttClient [▶ 55].
Connection to Azure IoT Hub cannot be established	Establishing the connection	Check whether the AWS IoT URL can be reached in principle from the system on which the Tc3_IotBase 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. 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_IotBase). If the Tc3_IotBase is used, the port configuration can be checked via the input parameter nHostPort at the function block FB_IotMqttClient [▶ 55].
The connection cannot be established after a CRL update. The variable eConnectionState shows the following value: MQTT_ERR_TLS_VERIFY_FAIL	Establishing the connection	The certificate of the message broker was revoked. In this case, please contact the operator of the message broker for further information.

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	Publish	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. Check whether your MQTT settings are valid and whether the Azure IoT Hub supports them. For further information consult the Beckhoff notes on establishing a connection to the Azure IoT Hub or the MSDN documentation.
Messages do not arrive at IBM Watson IoT	Publish	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. Check whether your MQTT settings are valid and whether IBM Watson IoT supports them. For further information consult the Beckhoff notes on establishing a connection to IBM Watson IoT or the IBM Watson documentation.
Messages content is not detected as a valid JSON message by the subscriber	Publish	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. 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.
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.
The connection to the message broker is suddenly interrupted in between.	Subscribe	Check if on the topics you subscribed to, a message exceeding the maximum message size (cMaxSizeOfMqttMessage [▶ 111]) may have been received. In this case, the MQTT client driver disconnects from the broker.

7.5 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.

Download finder

Our download finder contains all the files that we offer you for downloading. You will find application reports, technical documentation, technical drawings, configuration files and much more.

The downloads are available in various formats.

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 our internet page: 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
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
e-mail:	service@beckhoff.com

Beckhoff Headquarters

Beckhoff Automation GmbH & Co. KG

Huelshorstweg 20
33415 Verl
Germany

Phone:	+49 5246 963-0
e-mail:	info@beckhoff.com
web:	www.beckhoff.com

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