Manual

TC3 IoT Communication (MQTT)

TwinCAT 3

Version: 1.5
Date: 2018-12-13
Order No.: TF6701

BECKHOFF

Table of contents

1 Foreword

1.1 Notes on the documentation

This description is only intended for the use of trained specialists in control and automation engineering who are familiar with the applicable national standards.
It is essential that the documentation and the following notes and explanations are followed when installing and commissioning the components.
It is the duty of the technical personnel to use the documentation published at the respective time of each installation and commissioning.

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

Disclaimer

The documentation has been prepared with care. The products described are, however, constantly under development.
We reserve the right to revise and change the documentation at any time and without prior announcement.
No claims for the modification of products that have already been supplied may be made on the basis of the data, diagrams and descriptions in this documentation.

Trademarks

Beckhoff®, TwinCAT®, EtherCAT®, Safety over EtherCAT®, TwinSAFE®, XFC® and XTS® are registered trademarks of and licensed by Beckhoff Automation GmbH.
Other designations used in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owners.

Patent Pending

The EtherCAT Technology is covered, including but not limited to the following patent applications and patents:
EP1590927, EP1789857, DE102004044764, DE102007017835
with corresponding applications or registrations in various other countries.

The TwinCAT Technology is covered, including but not limited to the following patent applications and patents:
EP0851348, US6167425 with corresponding applications or registrations in various other countries.

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

Copyright

1.2 Safety instructions

Safety regulations

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

Exclusion of liability

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

Personnel qualification

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

Description of symbols

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

⚠ DANGER

Serious risk of injury!

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

⚠ WARNING

Risk of injury!

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

⚠ CAUTION

Personal injuries!

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

NOTE

Damage to the environment or devices

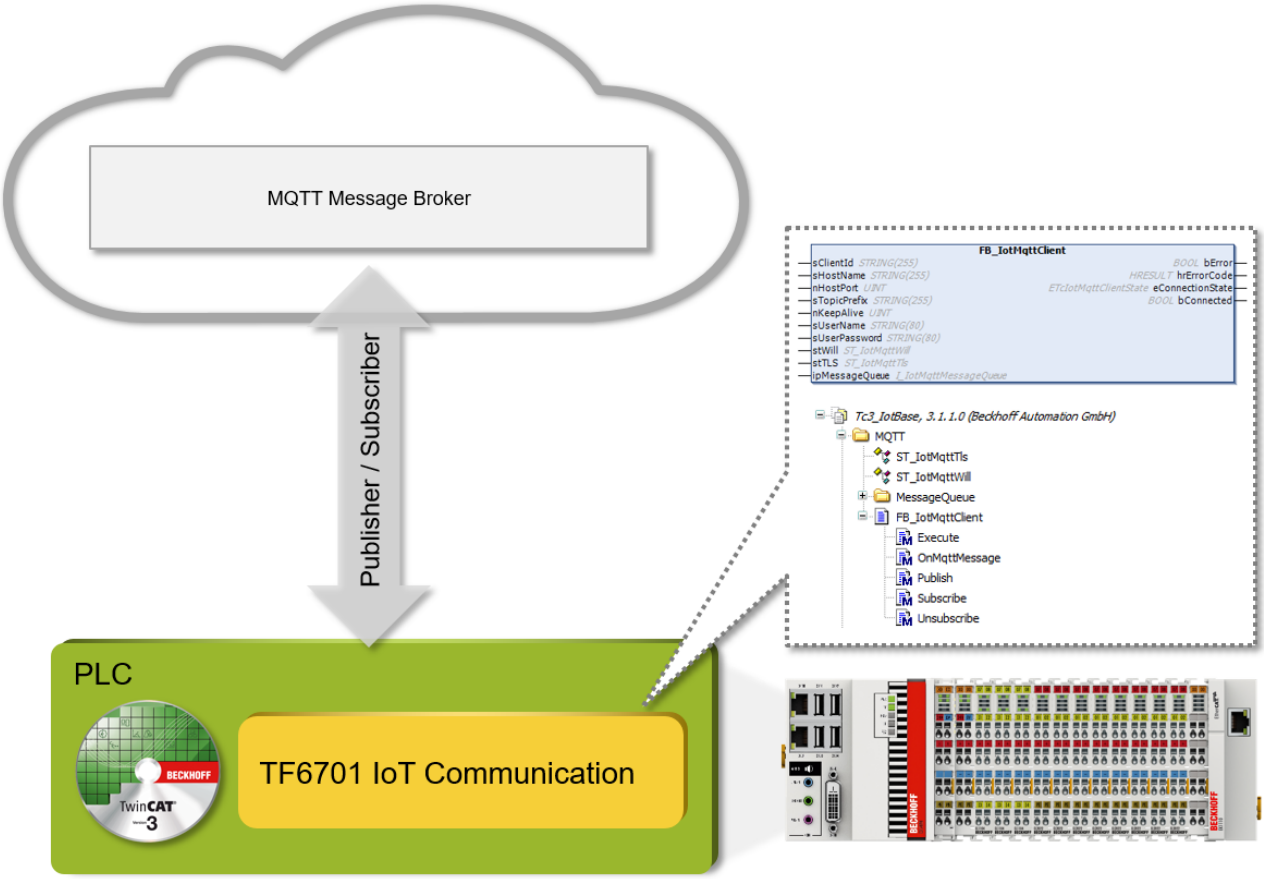
Failure to follow the instructions associated with this symbol can lead to damage to the environment or equipment.

Tip or pointer

This symbol indicates information that contributes to better understanding.

2 Overview

The function blocks of the PLC library Tc3_IotBase can be used for publisher/subscriber based data exchange between the local TwinCAT PLC and a message broker via the MQTT communication protocol. Symbols can be sent (publish mode) and received (subscribe mode).



Product components

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

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

3 Installation

3.1 System requirements

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

3.2 Installation

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

- TwinCAT XAE setup: Driver components and PLC library (Tc3_IotBase)
- TwinCAT XAR setup: Driver components

3.3 Licensing

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

The licensing of a TwinCAT 3 Function is described below. The description is divided into the following sections:

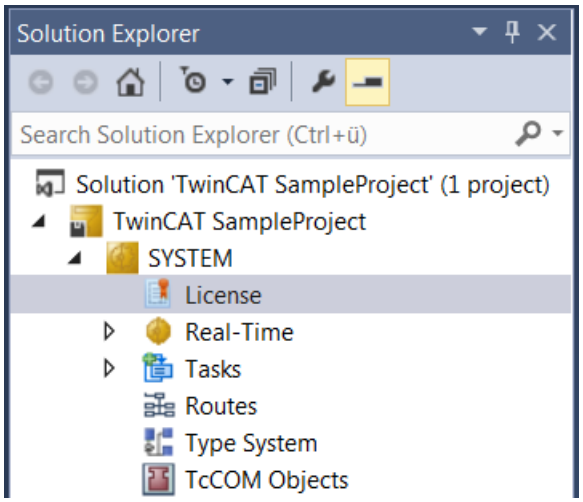
- Licensing a 7-day test version [▶ 8]
- Licensing a full version [▶ 10]

Further information on TwinCAT 3 licensing can be found in the “Licensing” documentation in the Beckhoff Information System (TwinCAT 3 > Licensing).

Licensing a 7-day test version

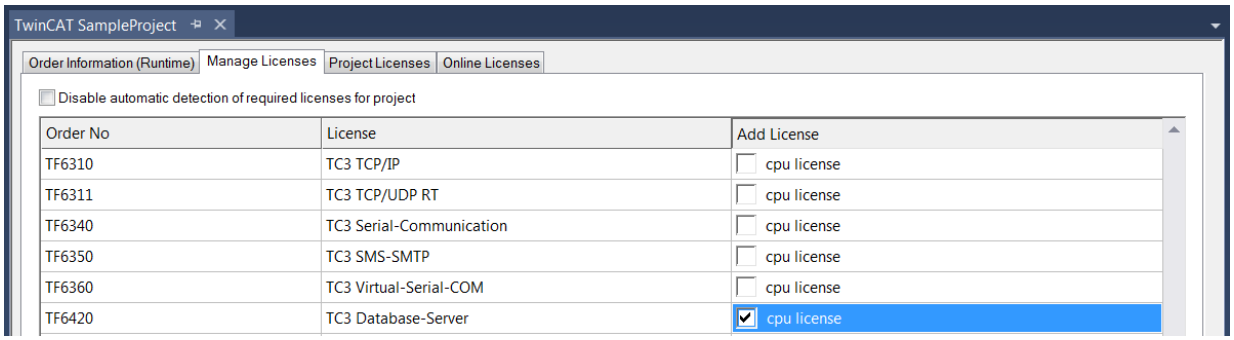
1. Start the TwinCAT 3 development environment (XAE).
2. Open an existing TwinCAT 3 project or create a new project.
3. If you want to activate the license for a remote device, set the desired target system. To do this, select the target system from the **Choose Target System** drop-down list in the 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.



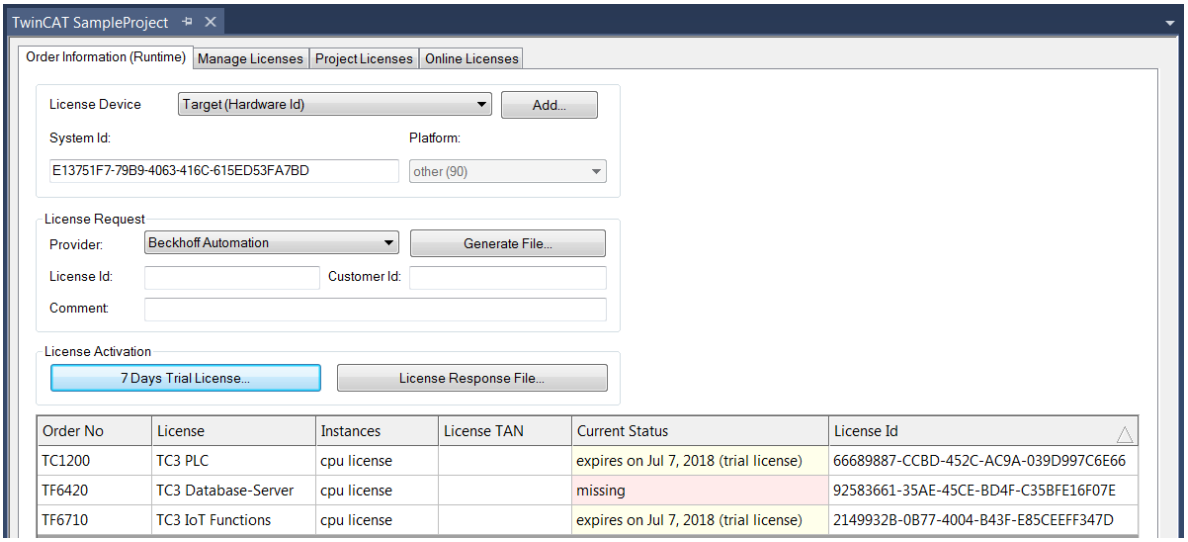
⇨ The TwinCAT 3 license manager opens.

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. “TF6420: TC3 Database Server“).

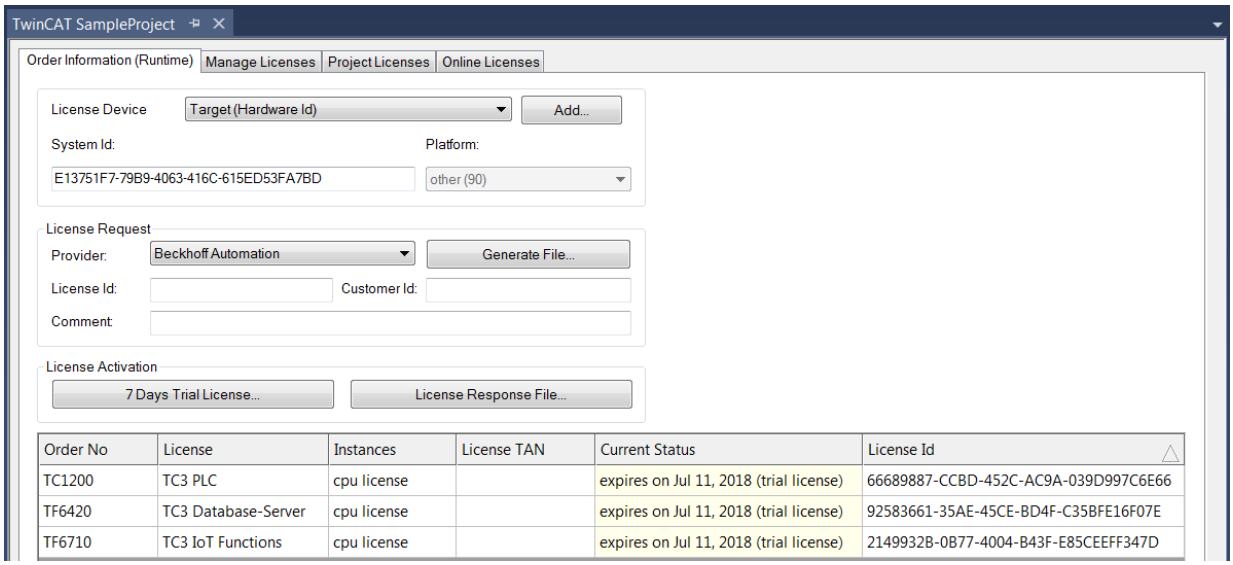


6. Open the **Order Information (Runtime)** tab.

⇨ 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 appears, confirm it and acknowledge the subsequent dialog indicating successful activation.

   ⇨ In the tabular overview of licenses, the license status now indicates the expiration date of the license.

9. Restart the TwinCAT system.

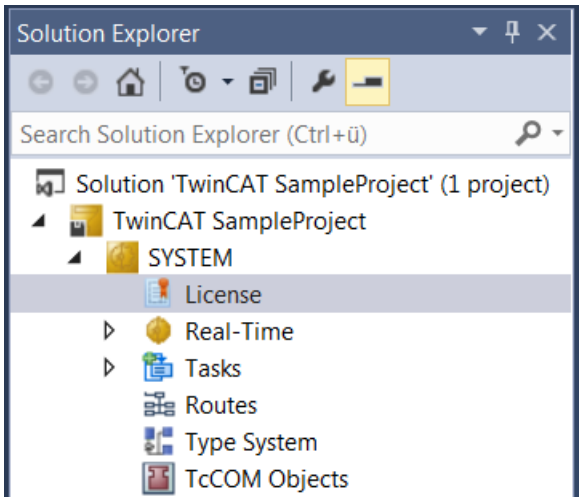
⇨ The 7-day trial version is enabled.

Licensing a full version

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.

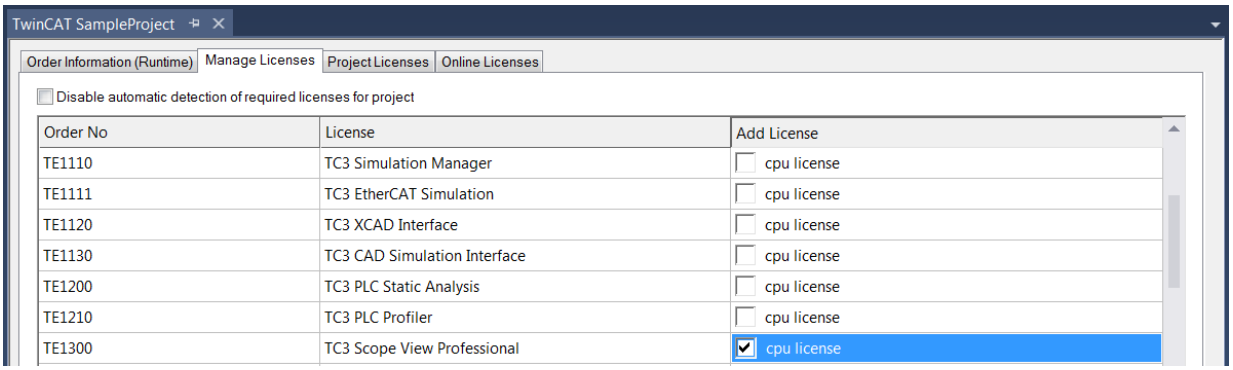
   ⇨ The licensing settings always refer to the selected target system. When the project is activated on the target system, the corresponding TwinCAT 3 licenses are automatically copied to this system.

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



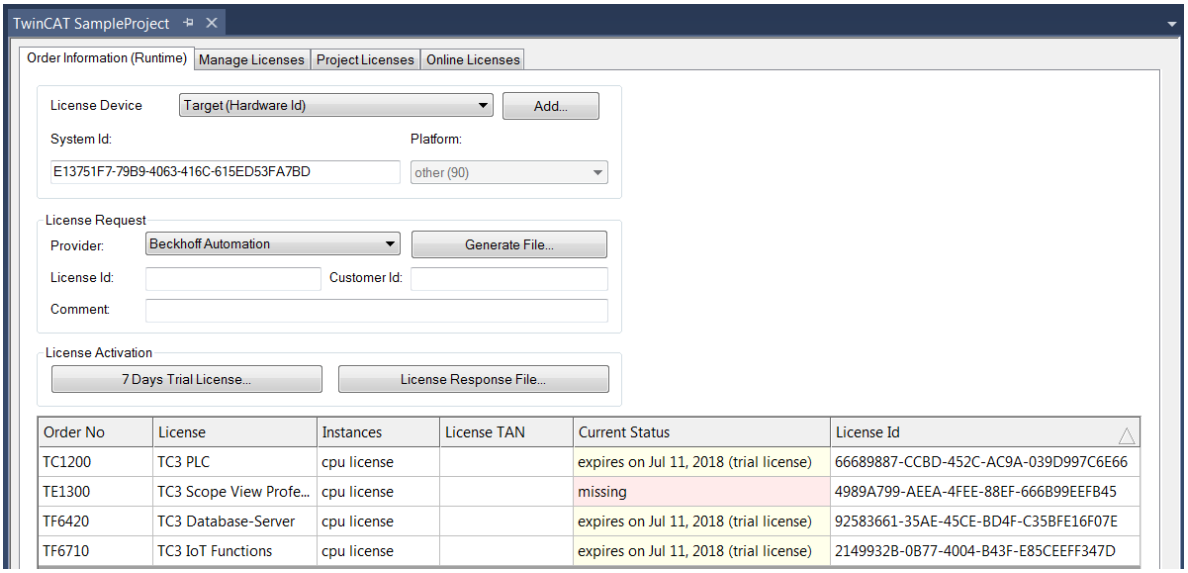
   ⇨ The TwinCAT 3 license manager opens.

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. “TE1300: TC3 Scope View Professional”).



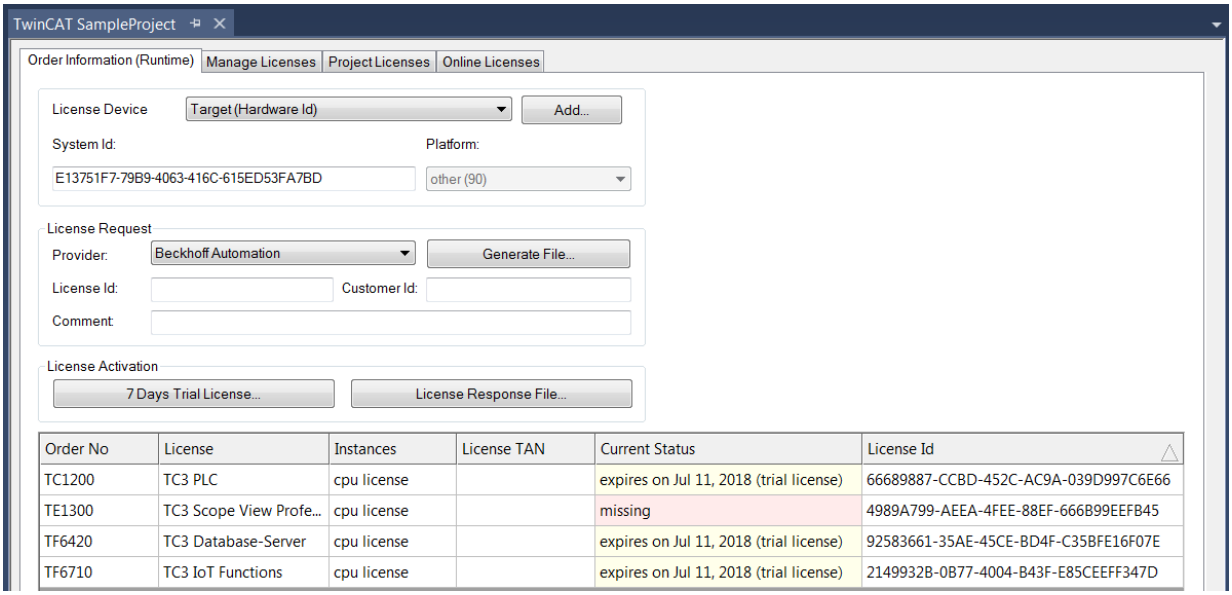
6. Open the **Order Information** tab.

   ⇨ In the tabular overview of licenses, the previously selected license is displayed with the status “missing”**.**

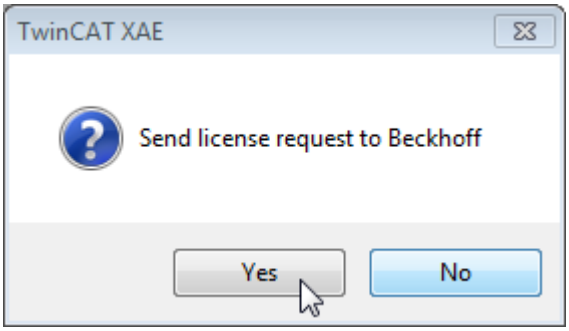


A TwinCAT 3 license is generally linked to two indices describing the platform to be licensed:
System ID: Uniquely identifies the device
Platform level: Defines the performance of the device
The corresponding **System Id** and **Platform** fields cannot be changed.

7. Enter the order number (**License Id**) for the license to be activated and optionally a separate order number (**Customer Id**), plus an optional comment for your own purposes (**Comment**). If you do not know your Beckhoff order number, please contact your Beckhoff sales contact.



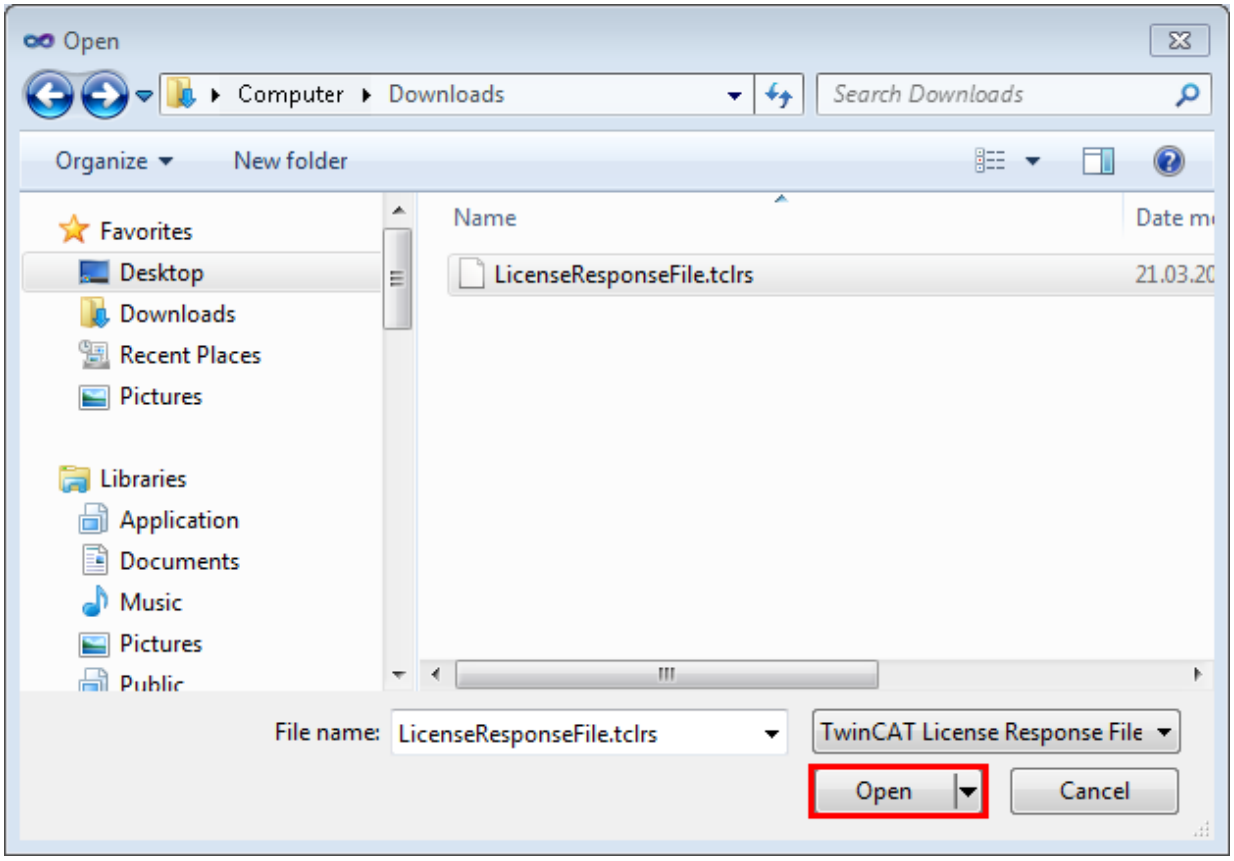
8. Click the **Generate File**... button to create a License Request File for the listed missing license.
   ⇨ A window opens, in which you can specify where the License Request File is to be stored. (We recommend accepting the default settings.)
9. Select a location and click **Save**.
   ⇨ A prompt appears asking whether you want to send the License Request File to the Beckhoff license server for verification:



- Click **Yes** to send the License Request File. A prerequisite is that an email program is installed on your computer and that your computer is connected to the internet. When you click **Yes**, the system automatically generates a draft email containing the License Request File with all the necessary information.
- Click **No** if your computer does not have an email program installed on it or is not connected to the internet. Copy the License Request File onto a data storage device (e.g. a USB stick) and send the file from a computer with internet access and an email program to the Beckhoff license server (tclicense@beckhoff.com) by email.

10. Send the License Request File.
    ⇨ The License Request File is sent to the Beckhoff license server. After receiving the email, the server compares your license request with the specified order number and returns a License Response File by email. The Beckhoff license server returns the License Response File to the same email address from which the License Request File was sent. The License Response File differs from the License Request File only by a signature that documents the validity of the license file content. You can view the contents of the License Response File with an editor suitable for XML files (e.g. “XML Notepad”). The contents of the License Response File must not be changed, otherwise the license file becomes invalid.
11. Save the License Response File.

12. To import the license file and activate the license, click **License Response File...** in the **Order Information** tab.
13. Select the License Response File in your file directory and confirm the dialog.



⇨ The License Response File is imported and the license it contains is activated. Existing demo licenses will be removed.

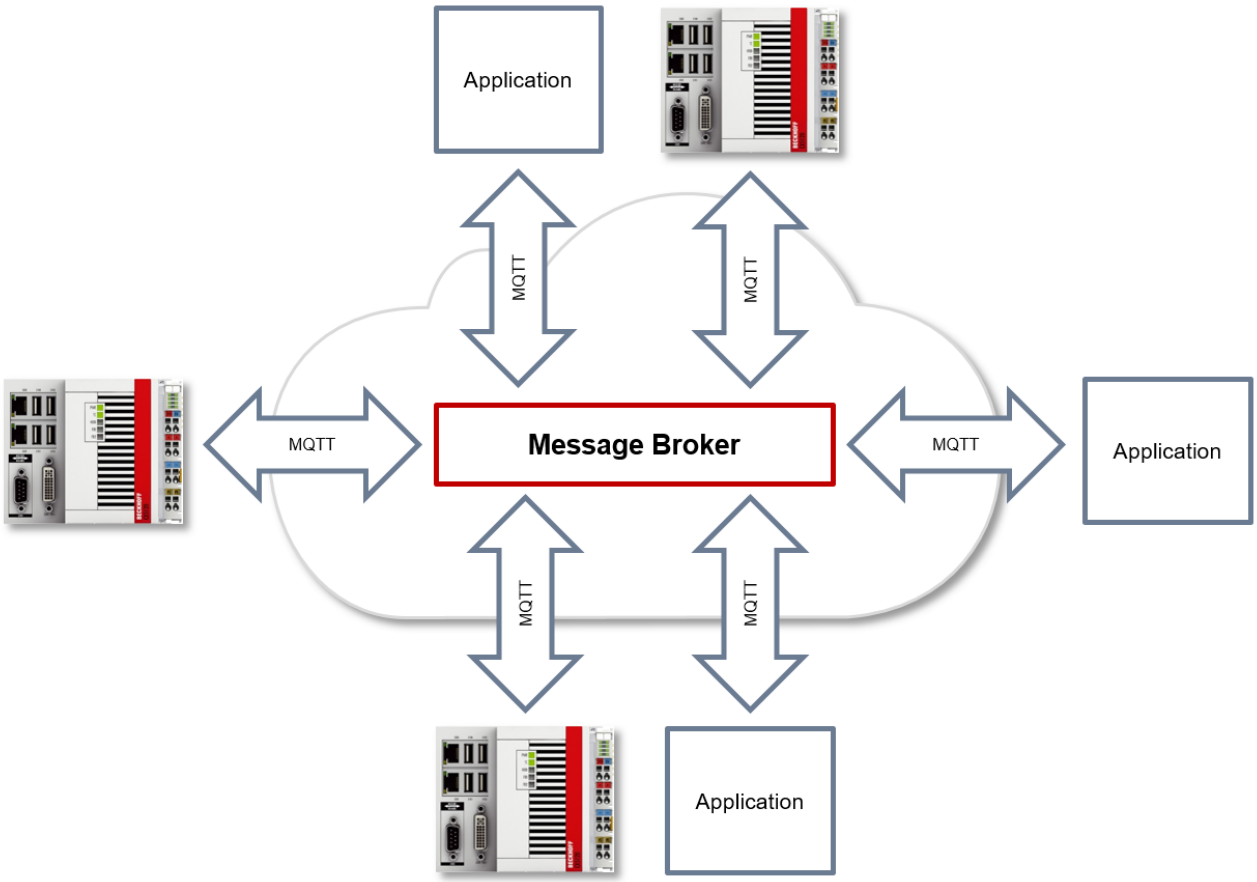
14. Restart the TwinCAT system.

⇨ The license becomes active when TwinCAT is restarted. The product can be used as a full version. During the TwinCAT restart the license file is automatically copied to the directory ...*TwinCAT\3.1\Target\License* on the respective target system.

4 Technical introduction

4.1 MQTT

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



Payload

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

Message payload formatting

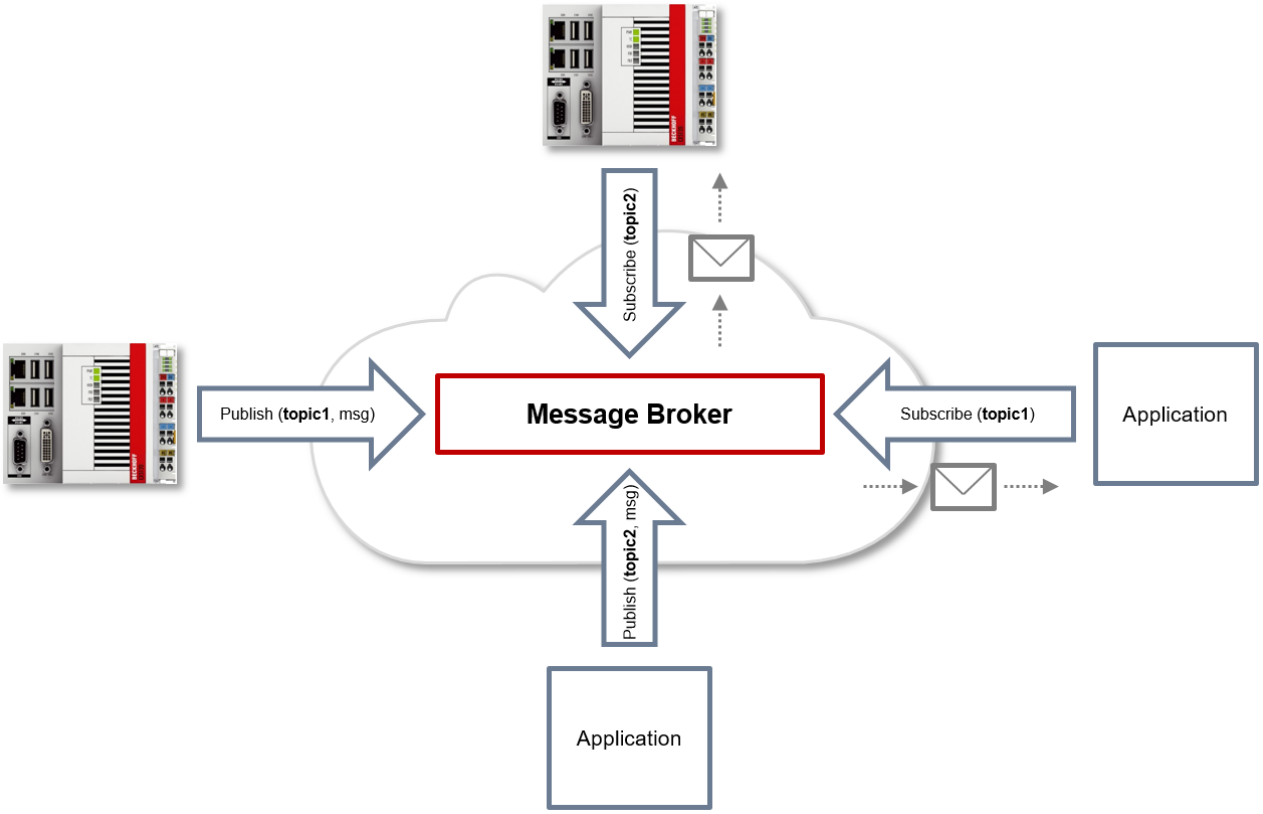
Note that the data type and the formatting of the content must be known to the sender and receiver side, particularly when binary information (alignment) or strings (with or without zero termination) are sent.

Topics

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

Example: Campus / Building1 / Floor2 / Room3 / Temperature

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



Communication example 1 from the diagram above:

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

Communication example 2 from the diagram above:

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

Wildcards

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

- Single-level wildcards
- Multi-level wildcards

Example for single-level wildcard:

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

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

Example for multi-level wildcard:

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

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

QoS (Quality of Service)

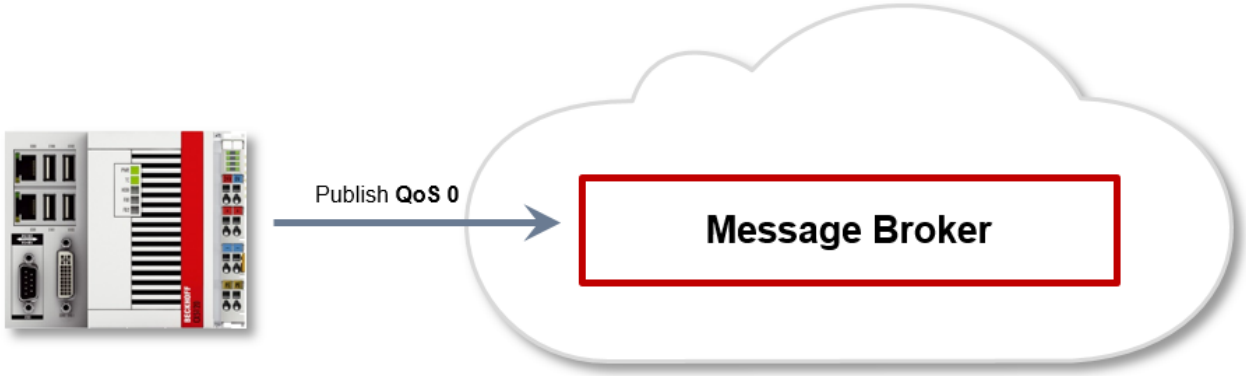
QoS is an arrangement between the sender and receiver of a message with regard to guaranteeing of the message transfer. MQTT features three different levels:

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

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

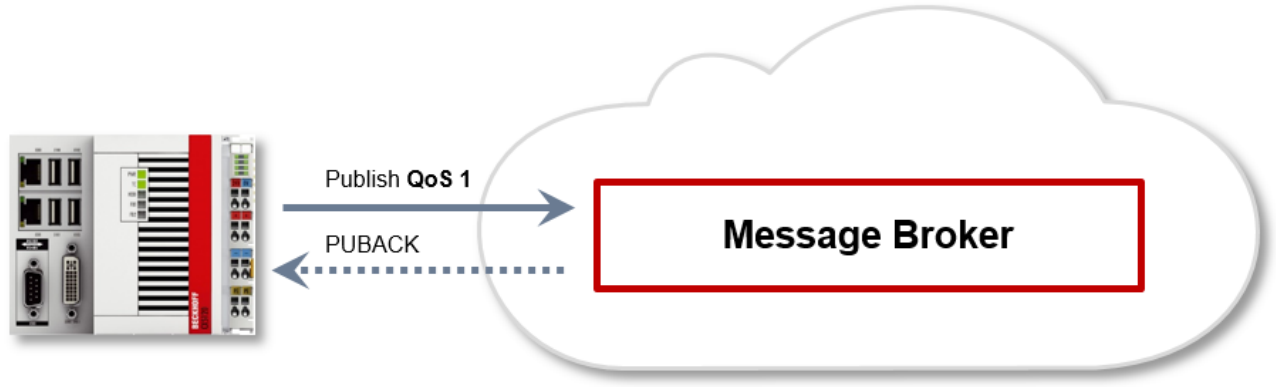
QoS-Level 0

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



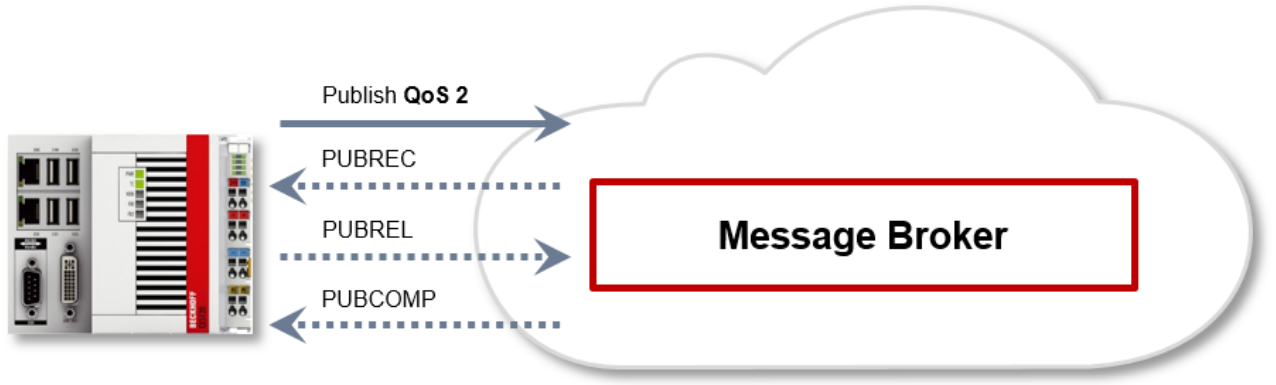
QoS-Level 1

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



QoS-Level 2

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



Security

When a connection to the message broker is established, it is possible to use security mechanisms [▶ 21] such as TLS, in order to encrypt the communication link or to realize authentication between client and message broker.

Sources

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

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

4.2 Application samples

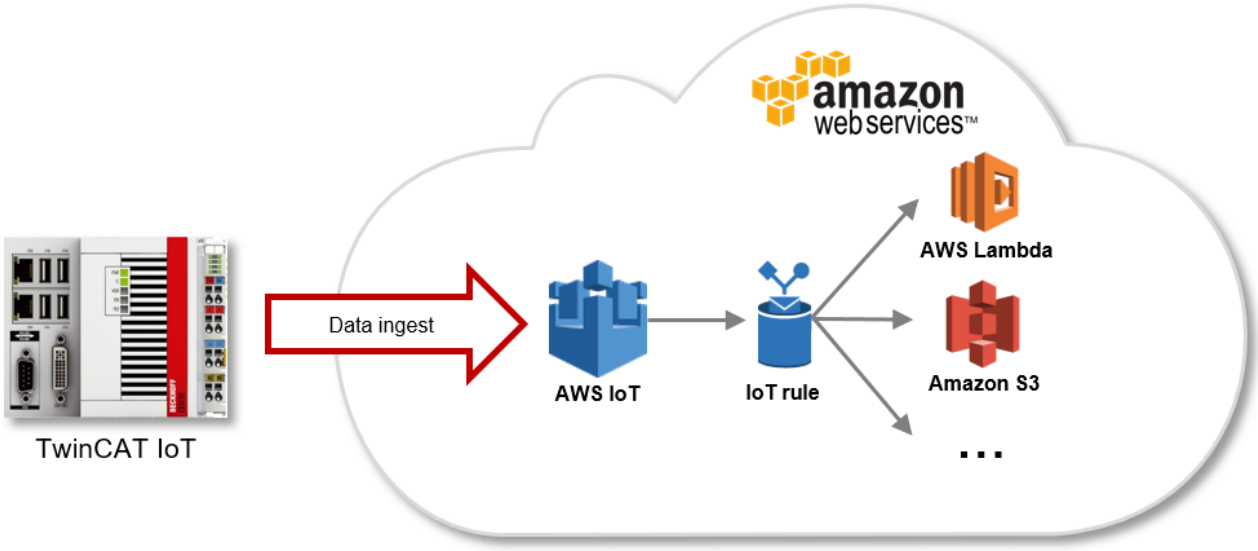
4.2.1 AWS IoT

AWS IoT is an IoT suite of the Amazon Web Services cloud, which offers several services for linking IoT devices with the AWS cloud and for processing incoming messages. From a device perspective, the services of AWS IoT enable simple and safe connection of IoT devices with AWS IoT by facilitating bidirectional communication between all enabled devices.

The use of TLS and the corresponding certificates is a prerequisite for establishing connections with AWS IoT. The corresponding certificates can be generated and downloaded from the AWS IoT management website. AWS IoT does not support QoS level 2. AWS IoT does not support retain messages.

TwinCAT IoT Communication (TF6701) configuration

Since the message broker of AWS IoT is based on the MQTT transport protocol, it is possible to use TF6701 IoT communication for sending messages to the broker or receiving messages from it. The following code snippet is based on the Tc3_IotBase PLC library and illustrates how a connection to AWS IoT can be established. Further information about AWS IoT and its configuration can be found on the AWS IoT website.



The following code snippet is based on the IotMqttSampleAwsIoT [▶ 112] sample and only shows the relevant part for establishing a connection to AWS IoT.

```
stMqttTls.sCA := 'c:\certs\root.pem';
stMqttTls.sCert := 'c:\certs\7613eee18a-certificate.pem.crt';
stMqttTls.sKeyFile := 'c:\certs\7613eee18a-private.pem.key';
fbMqttClient.sHostName:= 'aXXXXXXXX.iot.eu-west-1.amazonaws.com';
fbMqttClient.nHostPort:= 8883;
fbMqttClient.sClientId:= 'CX-12345';
fbMqttClient.stTLS := stMqttTls;
```

4.2.2 IBM Watson IoT

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

The topic structure for publish/subscribe is specified by IBM Watson IoT and cannot be changed.

TwinCAT IoT Communication (TF6701) configuration

Since IBM Watson IoT can be reached via the MQTT transport protocol, it is possible to use TF6701 IoT communication for sending messages to IBM Watson IoT or receiving messages from it. The following code snippet is based on the Tc3_IotBase PLC library and illustrates how a connection to IBM Watson IoT can be established. Further information about IBM Watson IoT and its configuration can be found on the IBM website.

The following code snippet is based on the IotMqttSampleIbmWatsonIoT [▶ 113] sample and only shows the relevant part for establishing a connection to IBM Watson IoT.

```
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
```

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.

4.2.3 MathWorks ThingSpeak

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

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

Functioning

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

Examples of possible actions on ThingSpeak™:

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

Applications

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

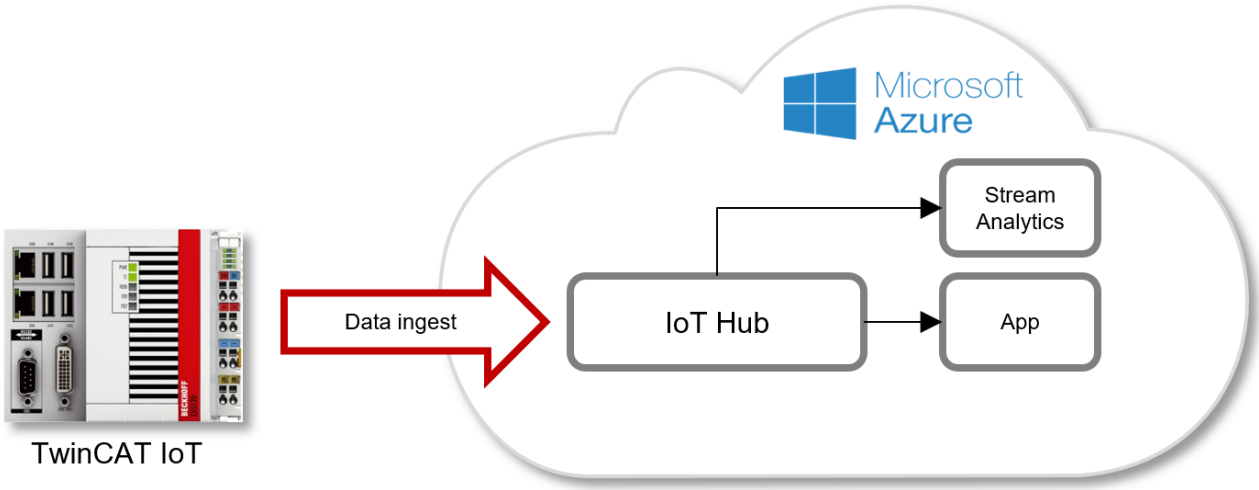
4.2.4 Microsoft Azure IoT Hub

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

The generation and use of an SAS token is a prerequisite for establishing connections with the Azure IoT Hub. This token can be generated with the Azure Device Explorer, for example. The Azure IoT Hub does not support QoS level 2. The Azure IoT Hub does not support retain messages. The topic structure for publish/subscribe is specified by the Azure IoT Hub and cannot be changed.

TwinCAT IoT Communication (TF6701) configuration

Since the Microsoft Azure IoT Hub can be reached via the MQTT transport protocol, it is possible to use TF6701 IoT communication for sending messages to the IoT Hub or receiving messages from it. The following code snippet is based on the Tc3_IotBase PLC library and illustrates how a connection to the Azure IoT Hub can be established. Further information on the Azure IoT Hub and its configuration can be found in the MSDN.



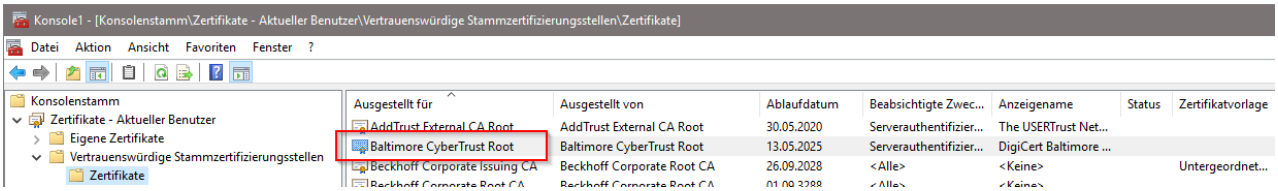
The following code snippet is based on the IotMqttSampleAzureIotHub [▶ 113] sample and only shows the relevant part for establishing a connection to the Azure IoT Hub.

```
// sTopicPub := 'devices/xxx/messages/events/readpipe/';
// sTopicSub := 'devices/xxx/messages/devicebound/#';
stTls.sCA := 'c:\TwinCAT\3.1\Config\Certificates\azure.crt';
stTls.sAzureSas := 'HostName=xxx.azure-
devices.net;DeviceId=CX-12345;SharedAccessSignature=SharedAccessSignature sr=xxx.azure-devices.net
%2fdevices%2fCX-12345&sig=123%3d&se=123';
fbMqttClient.stTLS := stTls;
```

The parameter sAzureSas of the structure ST_IotMqttTLS represents the SAS token of the Azure IoT Hub device, which can be generated via the Azure Device Explorer, for example. The stTls.sCA parameter is optional. If it is not specified, the path shown above is automatically used as CA certificate.

CA certificate

When establishing a connection to the Microsoft Azure IoT Hub via MQTT, the specification of a CA certificate is obligatory. The Baltimore Cyber Trust Root CA can be used as CA certificate. The associated public key can be extracted from the Microsoft Windows certificate console (Start > Run > mmc.exe, then add the "Certificates" snap-in). The Baltimore CA can then be found under the heading "Trusted Root Certification Authorities".

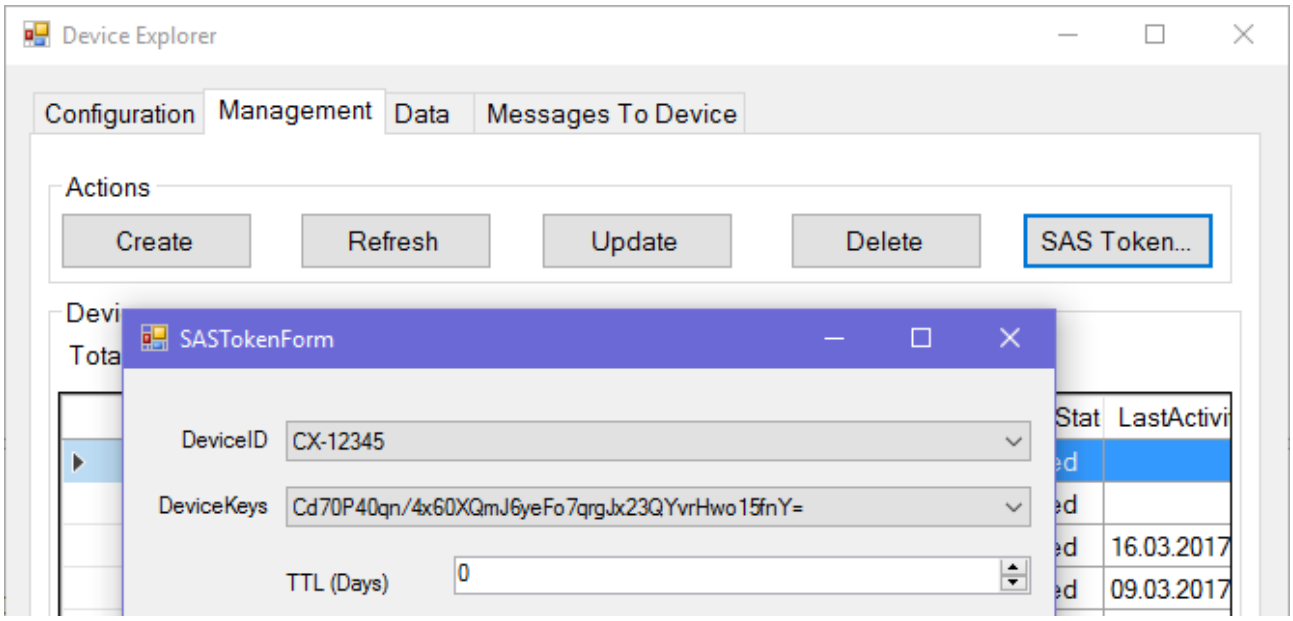


Publish

When data are published to the IoT Hub, the topic must be specified in the following form:

devices / *deviceId* / messages / events / readpipe

The device ID corresponds to the device ID of the registered device, as specified in the Device Explorer, for example.



Subscribe

For subscribing to data from the IoT Hub, the topic must be specified in the following form:

devices / *deviceId* / messages / devicebound / #

The device ID corresponds to the device ID of the registered device, as specified in the Device Explorer, for example.

4.3 Security

The MQTT specification offers MQTT clients the option to use user name/password authentication with the message broker. Common cryptography mechanisms such as TLS (Transport Layer Security) can be used to provide additional protection for the data communication between client and message broker. Depending on the message broker, different TLS mechanisms can be used:

- TLS-PreSharedKey (TSL-PSK)
- TLS-CertificateAuthority (TLS-CA)

TLS-PSK

The TLS PreSharedKey (PSK) method offers a simple option for realizing encryption between client and message broker. Client and broker recognize a common password, which is used to encrypt and decrypt the packages.

The following code snippet illustrates how TLS PSK can be used for TF6701, for example. This code snippet is also included in the IotMqttSampleTlsPsk [▶ 110] sample.

```
fbMqttClient.stTLS.sPskIdentity := 'my_Identity';
fbMqttClient.stTLS.aPskKey      := myPskKey;
fbMqttClient.stTLS.nPskKeyLen   := 15;
fbMqttClient.nHostPort          := 8883;
```

TLS-CA

Encryption and authentication via TLS can also be accomplished through a certificate authority (CA). The CA provides a signature via the public key for the message broker (the so-called server key) and usually also for all connecting clients. All communication devices can then trust each other, because the issuing certificate authority is trusted. Depending on the message broker, an MQTT client may connect without a dedicated client certificate. In this case the client simply uses the public key of the issuing certificate authority when it establishes a connection to the broker.

The following code snippet illustrates how TLS CA can be used for TF6701, for example. This code snippet is also included in the IotMqttSampleTlsCa [▶ 111] sample.

```
fbMqttClient.stTLS.sCA := 'C:\certs\ca.pem';
fbMqttClient.nHostPort := 8883;
```

The following code snippet illustrates how TLS CA with a client certificate can be used for TF6701, for example. This code snippet is also included in the IotMqttSampleTlsCa [▶ 111] sample.

```
fbMqttClient.stTLS.sCA   := 'C:\certs\ca.pem';
fbMqttClient.stTLS.sCert := 'C:\certs\client.cert';
fbMqttClient.nHostPort   := 8883;
```

4.4 JSON

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

Sample for a JSON object:

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

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

5 PLC API

5.1 Tc3_IotBase

5.1.1 FB_IotMqttClient

The function block enables communication with an MQTT broker.

A client function block is responsible for the connection to precisely one broker. The Execute() [▶ 25] 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 interna
lly)
    nKeepAlive    : UINT := 60;      // in seconds

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

    ipMessageFiFo : I_IotMqttMessageFiFo; // if received messages should be queued during call of Ex
ecute()
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. If no ID is specified, it is generated.
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 can be specified here. The default is 1883.
sTopicPrefix	STRING(255)	A topic prefix can be specified here that is appended automatically to 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.

Name	Type	Description
stWill	ST_IotMqttWill [▶ 28]	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 [▶ 29]	If the broker offers a TLS-secured connection, the required configuration can be implemented here.
ipMessageFiFo	I_IotMqttMessageFiFo	An instance of FB_IotMqttMessageQueue [▶ 30] 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 [▶ 106]) 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 [▶ 123] can be found in the Appendix.
eConnectionState	ETcIotMqttClientState	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 [▶ 25]	Method for background communication with the TwinCAT driver. The method must be called cyclically.
Publish [▶ 25]	Method for executing a publish operation to a MQTT message broker.
Subscribe [▶ 26]	Method for establishing a subscription.
Unsubscribe [▶ 27]	Method for removing a subscription.

Event-driven methods (callback methods)

Name	Description
OnMqttMessage [▶ 27]	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.

In order to be able to receive special characters and texts from a wide range of languages, the character set in the Tc3_IotBase library 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 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 Execute

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

Syntax

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

Inputs

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

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

5.1.1.2 Publish

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

Syntax

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

Inputs

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

Return value

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

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

Message payload formatting

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.

In order to be able to receive special characters and texts from a wide range of languages, the character set in the Tc3_IotBase library 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 library.

5.1.1.3 Subscribe

This method is used by the client to signal to the broker that it wants to receive all MQTT message with a particular topic.

Syntax

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

Inputs

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

Return value

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

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

5.1.1.4 Unsubscribe

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

Syntax

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

Inputs

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

Return value

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

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

5.1.1.5 OnMqttMessage

Callback method

This method must not be called by the user. Instead, the function block FB_IotMqttClient can be used to derive information and overwrite this method. While the Execute() method is called, the responsible TwinCAT driver can call the OnMqttMessage() method in the event of new incoming messages. In the event of several incoming messages the callback method is called several times, once per message. This must be taken into account when the method is implemented.
The application of the callback method is explained further in the sample IotMqttSampleUsingCallback [▶ 108].

Syntax

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

Inputs

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

Return value

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

5.1.2 ST_IotMqttWill

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

Syntax

Definition:

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

Parameter

Name	Type	Description
sTopic	STRING(255)	Message topic
pPayload	PVOID	Address for the payload of the message
nPayloadSize	UDINT	Size of the payload in bytes
eQoS	TcIotMqttQos	The “Quality of Service” parameter offers the following setting options: QoS level 0, QoS level 1, QoS level 2 (see QoS [▶ 16])
bRetain	BOOL	If bRetain is TRUE, the broker stores the message in order to make it available to subsequent subscribers.

Message payload formatting

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

5.1.3 ST_IotMqttTLS

TLS security setting for the MQTT client.

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

Syntax

Definition:

```
TYPE ST_IotMqttTls :
STRUCT
    sCA                : STRING(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'; // 'tlsv1' or 'tlsv1.1' or 'tlsv1.2' or 'tlsv1.3'
    bNoServerCertCheck : BOOL := FALSE;

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

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

Parameter

Name	Type	Description
sCA	STRING(255)	Certificate of the certificate authority (CA)
sCert	STRING(255)	Client certificate to be used for authentication at the broker
sKeyFile	STRING(255)	Private key of the client
sKeyPwd	STRING(255)	Password of the private key, if applicable
sCrl	STRING(255)	Path to the certificate revocation list, which may be present in PEM or DER format
sCiphers	STRING(255)	Ciper suites to be used, specified in OpenSSL string format
sVersion	STRING(80)	TLS version to be used
bNoServerCertCheck	BOOL	Disables verification of the server certificate validity
sPskIdentity	STRING(255)	PreSharedKey identity for TLS PSK connection
aPskKey	ARRAY[1..64] OF BYTE	PreSharedKey for TLS PSK connection
nPskKeyLen	USINT	Length of the PreSharedKey in bytes
sAzureSAS	STRING(511)	SAS token for connection to the Microsoft Azure IoT Hub [▶ 20]

5.1.4 Message Queue

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

5.1.4.1 FB_IotMqttMessageQueue

This function block offers a message queue for MQTT messages, which can be used with the block FB_IotMqttClient [▶ 23]. To this end an instance is declared and transferred at the input of FB_IotMqttClient. The function block operates based on the first in, first out principle (FIFO).
During the program sequence it is possible to check whether messages were collected in the message queue, and how many. The Dequeue() method is used for removing messages from the FIFO queue. The oldest message is output first.

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

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

Size of the MQTT message queue

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

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

Methods

Name	Description
Dequeue() [▶ 30]	Removes an MQTT message from the queue
ResetQueue() [▶ 31]	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

5.1.4.1.1 Dequeue

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

Syntax

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

Inputs

Name	Type	Description
fbMessage	REFERENCE TO FB_IotMqttMessage	For the message itself the reference is transferred to an instance of type FB_IotMqttMessage [▶ 31].

5.1.4.1.2 ResetQueue

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

5.1.4.2 FB_IotMqttMessage

If the TwintCAT MQTT client (FB_IotMqttClient [▶ 23]) is used in combination with a message queue (FB_IotMqttMessageQueue [▶ 30]), 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().

Methods

Name	Description
CompareTopic() [▶ 31]	Compares a specified topic with the topic in the MQTT message
GetTopic() [▶ 32]	Returns the topic of an MQTT message
GetPayload() [▶ 32]	Returns the content of an 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.

In order to be able to receive special characters and texts from a wide range of languages, the character set in the Tc3_IotBase library 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 library.

Requirements

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

5.1.4.2.1 CompareTopic

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

Syntax

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

Inputs

Name	Type	Description
sTopic	STRING	The topic to be compared is specified here.

5.1.4.2.2 GetTopic

Syntax

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

Inputs

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

5.1.4.2.3 GetPayload

Syntax

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

Inputs

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

5.2 Tc3_JsonXml

5.2.1 Function blocks

5.2.1.1 FB_JsonDomParser

5.2.1.1.1 AddArrayMember

This method adds an array member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.2 AddBase64Member

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

Syntax

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

Sample call:

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

5.2.1.1.3 AddBoolMember

This method adds a Bool member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.4 AddDateTimeMember

This method adds a DateTime member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.5 AddDcTimeMember

This method adds a DcTime member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.6 AddDoubleMember

This method adds a Double member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.7 AddFileTimeMember

This method adds a FileTime member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.8 AddHexBinaryMember

This method adds a HexBinary member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.9 AddInt64Member

This method adds an Int64 member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.10 AddIntMember

This method adds an Int member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.11 AddJsonMember

This method adds a JSON member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.12 AddNullMember

This method adds a NULL member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.13 AddObjectMember

This method adds an Object member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.14 AddStringMember

This method adds a String member to a JSON object.

Syntax

```
METHOD AddStringMember : SJsonValue
VAR_INPUT
  v : SJsonValue;
```

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

Sample call:

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

5.2.1.1.15 AddUint64Member

This method adds an UInt64 member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.16 AddUintMember

This method adds an UInt member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.17 ArrayBegin

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

Syntax

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

Sample call:

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

5.2.1.1.18 ArrayEnd

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

Syntax

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

Sample call:

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

5.2.1.1.19 ClearArray

This method deletes the content of an array.

Syntax

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

Sample call:

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

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

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

```
jsonDoc := fbJson.ParseDocument(sMessage);
jsonIterator := fbJson.MemberBegin(jsonDoc);
jsonIteratorEnd := fbJson.MemberEnd(jsonDoc);
WHILE jsonIterator <> jsonIteratorEnd DO
  sName := fbJson.GetMemberName(jsonIterator);
  jsonValue := fbJson.GetMemberValue(jsonIterator);
  IF sName = 'array' THEN
    jsonArrayIterator := fbJson.ArrayBegin(jsonValue);
    fbJson.ClearArray(jsonValue, jsonArrayIterator);
  END_IF
  jsonIterator := fbJson.NextMember(jsonIterator);
END_WHILE
```

5.2.1.1.20 CopyDocument

This method copies the contents of the JSON DOM memory into a variable of the data type STRING with an arbitrary length.

Syntax

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

Sample call:

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

5.2.1.1.21 CopyJson

This method extracts a JSON object from a property and saves it in a variable of the data type STRING.

Syntax

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

Sample call:

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

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

The value of the JSON object "clickTyp" is to be extracted and saved in a variable of the data type STRING. First of all, the JSON document is searched iteratively for the "clickType" property, after which its value is extracted by calling the CopyString() 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 = '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`

5.2.1.1.22 CopyString

This method extracts a string from a JSON property and saves it in a variable of the data type STRING.

Syntax

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

Sample call:

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

```
sMessage := ' {"serialNumber":"123","batteryVoltage":"1547mV","clickType":"SINGLE", "object":
{"Key1":42,"Key2":2,"Key3":3}}';
```

The value of the JSON property "object" is to be extracted and saved in a variable of the data type STRING. First of all, the JSON document is searched iteratively for the "object" property, after which all elements of the array are extracted by calling the CopyJson() 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 = 'object' 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:

```
{"Key1":42,"Key2":2,"Key3":3}
```

5.2.1.1.23 FindMember

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

Syntax

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

Sample call:

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

5.2.1.1.24 FindMemberPath

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

Syntax

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

Sample call:

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

5.2.1.1.25 GetArraySize

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

Syntax

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

Sample call:

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

5.2.1.1.26 GetArrayValue

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

Syntax

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

Sample call:

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

5.2.1.1.27 GetArrayValueByIdx

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

Syntax

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

Sample call:

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

5.2.1.1.28 GetBase64

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

Syntax

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

Sample call:

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

5.2.1.1.29 GetBool

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

Syntax

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

5.2.1.1.30 GetDateTime

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

Syntax

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

5.2.1.1.31 GetDcTime

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

Syntax

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

Sample call:

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

5.2.1.1.32 GetDocument

This method returns the content of the DOM memory as the data type STRING(255). The CopyDocument() method must be used with longer documents.

Syntax

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

Sample call:

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

5.2.1.1.33 GetDocumentLength

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

Syntax

```
METHOD GetDocumentLength: UDINT
```

Sample call:

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

5.2.1.1.34 GetDocumentRoot

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

Syntax

```
METHOD GetDocumentRoot : SJsonValue
```

Sample call:

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

5.2.1.1.35 GetDouble

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

Syntax

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

5.2.1.1.36 GetFileTime

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

Syntax

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

Sample call:

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

5.2.1.1.37 GetHexBinary

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

Syntax

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

Sample call:

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

5.2.1.1.38 GetInt

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

Syntax

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

5.2.1.1.39 GetInt64

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

Syntax

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

5.2.1.1.40 GetJson

This method returns the value of a property if this is in turn a JSON document.

Syntax

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

Sample call:

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

5.2.1.1.41 GetJsonLength

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

Syntax

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

Sample call:

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

5.2.1.1.42 GetMaxDecimalPlaces

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

Syntax

```
METHOD GetMaxDecimalPlaces : DINT
```

Sample call:

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

5.2.1.1.43 GetMemberName

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

Syntax

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

Sample call:

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

5.2.1.1.44 GetMemberValue

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

Syntax

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

Sample call:

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

5.2.1.1.45 GetString

This method returns the value of a property of the data type STRING(255).

Syntax

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

5.2.1.1.46 GetStringLength

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

Syntax

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

Sample call:

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

5.2.1.1.47 GetType

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

Syntax

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

TYPE EJsonType :
(
  eNullType := 0,
  eFalseType := 1,
  eTrueType := 2,
  eObjectType := 3,
  eArrayType := 4,
```

```
  eStringType := 5,
  eNumberType := 6
) DINT;
```

Sample call:

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

5.2.1.1.48 GetUint

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

Syntax

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

5.2.1.1.49 GetUint64

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

Syntax

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

5.2.1.1.50 HasMember

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

Syntax

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

Sample call:

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

5.2.1.1.51 IsArray

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

Syntax

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

5.2.1.1.52 IsBase64

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

Syntax

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

5.2.1.1.53 IsBool

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

Syntax

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

5.2.1.1.54 IsDouble

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

Syntax

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

5.2.1.1.55 IsFalse

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

Syntax

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

5.2.1.1.56 IsHexBinary

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

Syntax

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

Sample call:

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

5.2.1.1.57 IsInt

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

Syntax

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

5.2.1.1.58 IsInt64

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

Syntax

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

5.2.1.1.59 IsISO8601TimeFormat

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

Syntax

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

5.2.1.1.60 IsNull

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

Syntax

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

5.2.1.1.61 IsNumber

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

Syntax

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

5.2.1.1.62 IsObject

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

Syntax

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

5.2.1.1.63 IsString

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

Syntax

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

5.2.1.1.64 IsTrue

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

Syntax

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

5.2.1.1.65 IsUint

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

Syntax

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

5.2.1.1.66 IsUint64

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

Syntax

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

5.2.1.1.67 LoadDocumentFromFile

This method loads a JSON document from a file. A rising edge on the input parameter bExec triggers the loading procedure.

Syntax

```
METHOD LoadDocumentFromFile : BOOL
VAR_INPUT
  bExec : BOOL;
END_VAR
VAR_IN_OUT CONSTANT
  sFile : STRING;
END_VAR
```

Sample call:

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

5.2.1.1.68 MemberBegin

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

Syntax

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

Sample call:

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

5.2.1.1.69 MemberEnd

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

Syntax

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

Sample call:

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

5.2.1.1.70 NewDocument

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

Syntax

```
METHOD NewDocument : SJsonValue
```

Sample call:

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

5.2.1.1.71 NextArray

5.2.1.1.72 ParseDocument

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

Syntax

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

Sample call:

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

5.2.1.1.73 PushbackBase64Value

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

Syntax

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

Sample call:

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

5.2.1.1.74 PushbackBoolValue

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

Syntax

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

Sample call:

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

5.2.1.1.75 PushbackDateTimeValue

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

Syntax

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

Sample call:

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

5.2.1.1.76 PushbackDcTimeValue

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

Syntax

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

Sample call:

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

5.2.1.1.77 PushbackDoubleValue

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

Syntax

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

Sample call:

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

5.2.1.1.78 PushbackFileTimeValue

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

Syntax

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

Sample call:

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

5.2.1.1.79 PushbackHexBinaryValue

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

Syntax

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

Sample call:

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

5.2.1.1.80 PushbackInt64Value

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

Syntax

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

Sample call:

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

5.2.1.1.81 PushbackIntValue

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

Syntax

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

Sample call:

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

5.2.1.1.82 PushbackJsonValue

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

Syntax

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

Sample call:

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

5.2.1.1.83 PushbackNullValue

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

Syntax

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

Sample call:

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

5.2.1.1.84 PushbackStringValue

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

Syntax

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

Sample call:

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

5.2.1.1.85 PushbackUint64Value

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

Syntax

```
METHOD PushbackUint64Value : SJsonValue
VAR_INPUT
  v     : SJsonValue;
  value : ULINT;
END_VAR
```

Sample call:

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

5.2.1.1.86 PushbackUintValue

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

Syntax

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

Sample call:

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

5.2.1.1.87 RemoveAllMembers

This method removes all child elements from a given property.

Syntax

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

Sample call:

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

5.2.1.1.88 RemoveArray

This method deletes the value of the current array iterator.

Syntax

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

Sample call:

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

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

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

```
jsonDoc := fbJson.ParseDocument(sMessage);
jsonIterator := fbJson.MemberBegin(jsonDoc);
jsonIteratorEnd := fbJson.MemberEnd(jsonDoc);
WHILE jsonIterator <> jsonIteratorEnd DO
  sName := fbJson.GetMemberName(jsonIterator);
  jsonValue := fbJson.GetMemberValue(jsonIterator);
  IF sName = 'array' THEN
    jsonArrayIterator := fbJson.ArrayBegin(jsonValue);
    fbJson.RemoveArray(jsonValue, jsonArrayIterator);
  END_IF
  jsonIterator := fbJson.NextMember(jsonIterator);
END_WHILE
```

5.2.1.1.89 RemoveMember

This method deletes the property at the current iterator.

Syntax

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

Sample call:

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

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

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

```
jsonDoc := fbJson.ParseDocument(sMessage);
jsonIterator := fbJson.MemberBegin(jsonDoc);
jsonIteratorEnd := fbJson.MemberEnd(jsonDoc);
WHILE jsonIterator <> jsonIteratorEnd DO
  sName := fbJson.GetMemberName(jsonIterator);
  IF sName = 'array' THEN
    fbJson.RemoveMember(jsonDoc, jsonIterator);
  END_IF
  jsonIterator := fbJson.NextMember(jsonIterator);
END_WHILE
```

5.2.1.1.90 RemoveMemberByName

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

Syntax

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

Sample call:

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

5.2.1.1.91 SaveDocumentToFile

This method saves a JSON document in a file.

Syntax

```
METHOD SaveDocumentToFile : BOOL
VAR_INPUT
  bExec : REFERENCE TO BOOL;
END_VAR
VAR_IN_OUT CONSTANT
    sFile : STRING;
END_VAR
```

Sample call:

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

5.2.1.1.92 SetArray

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

Syntax

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

Sample call:

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

5.2.1.1.93 SetBase64

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

Syntax

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

Sample call:

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

5.2.1.1.94 SetBool

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

Syntax

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

Sample call:

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

5.2.1.1.95 SetDateTime

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

Syntax

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

Sample call:

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

5.2.1.1.96 SetDcTime

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

Syntax

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

Sample call:

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

5.2.1.1.97 SetDouble

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

Syntax

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

Sample call:

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

5.2.1.1.98 SetFileTime

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

Syntax

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

Sample call:

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

5.2.1.1.99 SetHexBinary

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

Syntax

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

Sample call:

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

5.2.1.1.100 SetInt

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

Syntax

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

Sample call:

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

5.2.1.1.101 SetInt64

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

Syntax

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

Sample call:

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

5.2.1.1.102 SetJson

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

Syntax

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

Sample call:

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

5.2.1.1.103 SetMaxDecimalPlaces

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

Syntax

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

Sample call:

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

5.2.1.1.104 SetNull

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

Syntax

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

Sample call:

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

5.2.1.1.105 SetObject

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

Syntax

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

Sample call:

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

5.2.1.1.106 SetString

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

Syntax

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

Sample call:

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

5.2.1.1.107 SetUint

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

Syntax

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

Sample call:

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

5.2.1.1.108 SetUint64

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

Syntax

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

Sample call:

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

5.2.1.2 FB_JsonSaxReader

5.2.1.2.1 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
```

Sample call:

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

5.2.1.2.2 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;
END_VAR
```

Sample call:

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

5.2.1.2.3 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;
END_VAR
```

Sample call:

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

5.2.1.2.4 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;
END_VAR
```

Sample call:

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

5.2.1.2.5 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
```

Sample call:

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

5.2.1.2.6 GetLastParseResult

Syntax

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

5.2.1.2.7 IsBase64

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

Syntax

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

Sample call:

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

5.2.1.2.8 IsHexBinary

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

Syntax

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

Sample call:

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

5.2.1.2.9 IsISO8601TimeFormat

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

Syntax

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

Sample call:

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

5.2.1.2.10 Parse

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

Syntax

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

5.2.1.2.11 ParseValues

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

Syntax

```
METHOD ParseValues : BOOL
VAR_IN_OUT CONSTANT
  sJson : STRING;
END_VAR
VAR_INPUT
  ipHdl : ITcJsonSaxValues;
END_VAR
```

5.2.1.3 FB_JsonSaxWriter

5.2.1.3.1 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() [▶ 65].

Syntax

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

5.2.1.3.2 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() [▶ 65].

Syntax

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

Sample call:

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

5.2.1.3.3 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() [▶ 65].

Syntax

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

Sample call:

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

5.2.1.3.4 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() [▶ 65].

Syntax

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

Sample call:

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

5.2.1.3.5 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() [▶ 65].

Syntax

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

Sample call:

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

5.2.1.3.6 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() [▶ 65].

Syntax

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

Sample call:

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

5.2.1.3.7 AddHexBinary

This method adds a hex binary value to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 65].

Syntax

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

Sample call:

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

5.2.1.3.8 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 [▶ 63], AddBool [▶ 64], AddDateTime [▶ 64], AddDcTime [▶ 64], AddDint [▶ 64], AddFileTime [▶ 65], AddHexBinary [▶ 65], AddLint [▶ 67], AddLreal [▶ 68], AddNull [▶ 68], AddRawArray [▶ 68], AddRawObject [▶ 68], AddReal [▶ 69], AddString [▶ 69], AddUdint [▶ 69], AddUlint [▶ 69].

Syntax

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

Sample call:

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

5.2.1.3.9 AddKeyBool

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

Syntax

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

Sample call:

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

5.2.1.3.10 AddKeyDateTime

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

Syntax

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

Sample call:

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

5.2.1.3.11 AddKeyDcTime

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

Syntax

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

Sample call:

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

5.2.1.3.12 AddKeyFileTime

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

Syntax

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

Sample call:

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

5.2.1.3.13 AddKeyLreal

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

Syntax

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

Sample call:

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

5.2.1.3.14 AddKeyNull

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

Syntax

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

Sample call:

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

5.2.1.3.15 AddKeyNumber

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

Syntax

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

Sample call:

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

5.2.1.3.16 AddKeyString

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

Syntax

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

Sample call:

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

5.2.1.3.17 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() [▶ 65].

Syntax

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

Sample call:

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

5.2.1.3.18 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() [▶ 65].

Syntax

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

Sample call:

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

5.2.1.3.19 AddNull

This method adds the value zero to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 65].

Syntax

```
METHOD AddNull
```

Sample call:

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

5.2.1.3.20 AddRawArray

This method adds a valid JSON array to a given property as a value. The array to be added must be in a valid JSON format and may only be added if the SAX writer is at a correspondingly valid position, i.e. for example, directly after a preceding AddKey() [▶ 65], StartArray() [▶ 71] or as the first call after a ResetDocument() [▶ 71].

Syntax

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

Sample call:

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

5.2.1.3.21 AddRawObject

This method adds a valid JSON object to a given property as a value. The object to be added must be in a valid JSON format and may only be added if the SAX writer is at a correspondingly valid position, i.e. for example, directly after a preceding AddKey() [▶ 65], StartArray() [▶ 71] or as the first call after a ResetDocument() [▶ 71].

Syntax

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

Sample call:

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

5.2.1.3.22 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() [▶ 65].

Syntax

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

Sample call:

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

5.2.1.3.23 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() [▶ 65].

Syntax

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

Sample call:

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

5.2.1.3.24 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() [▶ 65].

Syntax

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

Sample call:

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

5.2.1.3.25 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() [▶ 65].

Syntax

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

Sample call:

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

5.2.1.3.26 CopyDocument

This method copies the contents of the JSON object currently created with the SAX writer into a target variable of the data type STRING.

Syntax

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

Sample call:

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

5.2.1.3.27 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()
```

Sample call:

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

5.2.1.3.28 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
```

Sample call:

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

5.2.1.3.29 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).

Syntax

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

Sample call:

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

5.2.1.3.30 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
```

Sample call:

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

5.2.1.3.31 GetMaxDecimalPlaces

5.2.1.3.32 ResetDocument

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

Syntax

```
METHOD ResetDocument
```

Sample call:

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

5.2.1.3.33 SetMaxDecimalPlaces

5.2.1.3.34 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()
```

Sample call:

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

5.2.1.3.35 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
```

Sample call:

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

5.2.1.4 FB_JsonReadWriteDataType

5.2.1.4.1 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 [▶ 63] 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
```

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 [▶ 115].

Sample call:

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

5.2.1.4.2 AddJsonKeyValueFromSymbol

This method generates the JSON representation of a PLC data structure on an FB_JsonSaxWriter [▶ 63] 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() [▶ 72], 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
```

A complete sample of how to use this method can be found in section Tc3JsonXmlSampleJsonDataType [▶ 115].

Sample call:

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

5.2.1.4.3 AddJsonValueFromSymbol

This method generates the JSON representation of a PLC data structure on an FB_JsonSaxWriter [▶ 63] 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
```

A complete sample of how to use this method can be found in section Tc3JsonXmlSampleJsonDataType [▶ 115].

Sample call:

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

5.2.1.4.4 GetDataTypeNameByAddress

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

Syntax

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

Sample call:

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

5.2.1.4.5 GetJsonFromSymbol

This method generates the corresponding JSON representation of a symbol. In contrast to the AddJsonValueFromSymbol() [▶ 72] 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
```

Input parameter nJson

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

Sample call:

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

5.2.1.4.6 GetJsonStringFromSymbol

This method generates the corresponding JSON representation of a symbol. In contrast to the AddJsonValueFromSymbol() [▶ 72] 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.

Syntax

```
METHOD GetJsonStringFromSymbol : STRING(16384)
VAR_IN_OUT CONSTANT
  sDatatype : STRING;
END_VAR
VAR_INPUT
  nData : UDINT;
  pData : PVOID;
END_VAR
```

Sample call:

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

5.2.1.4.7 GetJsonStringFromSymbolProperties

This method generates a corresponding JSON representation of PLC attributes on a symbol. In contrast to the AddJsonKeyPropertiesFromSymbol [▶ 71] 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.

Syntax

```
METHOD GetJsonStringFromSymbolProperties : STRING(16384)
VAR_IN_OUT CONSTANT
  sDatatype : STRING;
  sProperties : STRING;
END_VAR
```

Sample call:

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

5.2.1.4.8 GetSymbolNameByAddress

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

Syntax

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

Sample call:

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

5.2.1.4.9 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
```

Sample call:

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

5.2.1.5 FB_XmlDomParser

5.2.1.5.1 AppendAttribute

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

Syntax

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

Sample call:

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

5.2.1.5.2 AppendAttributeAsBool

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

Syntax

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

Sample call:

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

5.2.1.5.3 AppendAttributeAsDouble

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

Syntax

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

Sample call:

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

5.2.1.5.4 AppendAttributeAsFloat

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

Syntax

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

Sample call:

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

5.2.1.5.5 AppendAttributeAsInt

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

Syntax

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

Sample call:

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

5.2.1.5.6 AppendAttributeAsLint

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

Syntax

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

Sample call:

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

5.2.1.5.7 AppendAttributeAsUint

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

Syntax

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

Sample call:

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

5.2.1.5.8 AppendAttributeAsUlint

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

Syntax

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

Sample call:

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

5.2.1.5.9 AppendAttributeCopy

This method adds a new attribute to an existing node. The name and value of the new attribute are copied from an existing attribute. The existing attribute is transferred to the method as input parameter.

Syntax

```
METHOD AppendAttributeCopy : SXmlAttribute
INPUT_VAR
  n : SXmlNode;
  copy : SXmlAttribute;
END_VAR
```

Sample call:

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

5.2.1.5.10 AppendChild

This method inserts a new node below an existing node. The value of the new node has the data type STRING. The name and value of the new node and a reference to the existing node are transferred to the method as input parameters. The method returns a reference to the newly added node. The input parameter cdata indicates whether the value of the node is to be encapsulated in a CDATA function block, so that certain special characters such as "<" and ">" are allowed as values.

Syntax

```
METHOD AppendChild : SXmlNode
VAR_INPUT
  n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name : STRING;
  value : STRING;
END_VAR
VAR_INPUT
  cdata : BOOL;
END_VAR
```

Sample call:

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

5.2.1.5.11 AppendChildAsBool

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

Syntax

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

Sample call:

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

5.2.1.5.12 AppendChildAsDouble

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

Syntax

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

Sample call:

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

5.2.1.5.13 AppendChildAsFloat

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

Syntax

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

Sample call:

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

5.2.1.5.14 AppendChildAsInt

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

Syntax

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

Sample call:

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

5.2.1.5.15 AppendChildAsLint

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

Syntax

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

Sample call:

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

5.2.1.5.16 AppendChildAsUint

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

Syntax

```
METHOD AppendChildAsUint : SXmlNode
VAR_INPUT
  n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name : STRING;
END_VAR
VAR_INPUT
  value : UDINT;
END_VAR
```

Sample call:

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

5.2.1.5.17 AppendChildAsUlint

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

Syntax

```
METHOD AppendChildAsUlint : SXmlNode
VAR_INPUT
  n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name : STRING;
END_VAR
VAR_INPUT
  value : ULINT;
END_VAR
```

Sample call:

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

5.2.1.5.18 AppendCopy

This method inserts a new node below an existing node. The name and value of the new node are copied from an existing node. The references to the existing nodes are transferred to the method as input parameters. The method returns a reference to the newly added node.

Syntax

```
METHOD AppendCopy : SXmlNode
VAR_INPUT
  n : SXmlNode;
  copy : SXmlNode;
END_VAR
```

Sample call:

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

5.2.1.5.19 AppendNode

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

Syntax

```
METHOD AppendNode : SXmlNode
VAR_INPUT
  n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name : STRING;
END_VAR
```

Sample call:

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

5.2.1.5.20 Attributes

This method can be used to read the attribute of a given XML node. The XML node and the name of the attribute are transferred to the method as input parameters. After the method has been called, further methods have to be called, for example to read the value of the attribute, e.g. AttributeAsInt().

Syntax

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

Sample call:

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

5.2.1.5.21 AttributeAsBool

This method returns the value of an attribute as data type Boolean. The attribute is transferred to the method as input parameter.

Syntax

```
METHOD AttributeAsBool : BOOL
VAR_INPUT
  a : SXmlAttribute;
END_VAR
```

Sample call:

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

5.2.1.5.22 AttributeAsDouble

This method returns the value of an attribute as data type Double. The attribute is transferred to the method as input parameter.

Syntax

```
METHOD AttributeAsDouble : LREAL
VAR_INPUT
  a : SXmlAttribute;
END_VAR
```

Sample call:

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

5.2.1.5.23 AttributeAsFloat

This method returns the value of an attribute as data type Float. The attribute is transferred to the method as input parameter.

Syntax

```
METHOD AttributeAsFloat : REAL
VAR_INPUT
  a : SXmlAttribute;
END_VAR
```

Sample call:

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

5.2.1.5.24 AttributeAsInt

This method returns the value of an attribute as a data type Integer. The attribute is transferred to the method as input parameter.

Syntax

```
METHOD AttributeAsInt : DINT
VAR_INPUT
  a : SXmlAttribute;
END_VAR
```

Sample call:

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

5.2.1.5.25 AttributeAsLint

This method returns the value of an attribute as a data type Integer64. The attribute is transferred to the method as input parameter.

Syntax

```
METHOD AttributeAsLint : LINT
VAR_INPUT
  a : SXmlAttribute;
END_VAR
```

Sample call:

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

5.2.1.5.26 AttributeAsUint

This method returns the value of an attribute as data type Unsigned Integer. The attribute is transferred to the method as input parameter.

Syntax

```
METHOD AttributeAsUint : UDINT
VAR_INPUT
  a : SXmlAttribute;
END_VAR
```

Sample call:

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

5.2.1.5.27 AttributeAsUlint

This method returns the value of an attribute as data type Unsigned Integer64. The attribute is transferred to the method as input parameter.

Syntax

```
METHOD AttributeAsUlint : ULINT
VAR_INPUT
  a : SXmlAttribute;
END_VAR
```

Sample call:

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

5.2.1.5.28 AttributeBegin

This method returns an iterator over all attributes of an XML node. The XML node is transferred to the method as input parameter.

Syntax

```
METHOD AttributeBegin : SXmlIterator
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Sample call:

```
xmlIterator := fbXml.AttributeBegin(xmlNode);
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlAttr := fbXml.AttributeFromIterator(xmlIterator);
  nAttrValue := fbXml.AttributeAsInt(xmlAttr);
  xmlIterator := fbXml.Next(xmlIterator);
END_WHILE
```

5.2.1.5.29 AttributeFromIterator

This method converts the current position of an iterator to an XML attribute object. The iterator is transferred to the method as input parameter.

Syntax

```
METHOD AttributeFromIterator : SXmlAttribute
VAR_INPUT
  it : SXmlIterator;
END_VAR
```

Sample call:

```
xmlIterator := fbXml.AttributeBegin(xmlNode);
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlAttr := fbXml.AttributeFromIterator(xmlIterator);
  nAttrValue := fbXml.AttributeAsInt(xmlAttr);
  xmlIterator := fbXml.Next(xmlIterator);
END_WHILE
```

5.2.1.5.30 AttributeName

This method returns the name of a given attribute. The attribute is transferred to the method as input parameter.

Syntax

```
METHOD AttributeName : STRING
VAR_INPUT
  a : SXmlAttribute;
END_VAR
```

Sample call:

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

5.2.1.5.31 Attributes

This method is used to navigate through the DOM and returns an iterator for all attributes found at an XML node. The iterator can then be used for further navigation through the elements that were found. The node and a reference to the iterator are transferred to the method as input parameters.

Syntax

```
METHOD Attributes : SXmlAttribute
VAR_INPUT
  n : SXmlNode;
  it : REFERENCE TO SXmlIterator;
END_VAR
```

Sample call:

```
xmlRet := fbXml.Attributes(xmlNode, xmlIterator);
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlMachineAttrRef := fbXml.Attribute(xmlIterator);
  xmlMachineAttrText := fbXml.AttributeText(xmlMachineAttrRef);
  xmlIterator := fbXml.Next(xmlIterator);
END_WHILE
```

5.2.1.5.32 AttributeText

This method returns the text of a given attribute. The attribute is transferred to the method as input parameter.

Syntax

```
METHOD AttributeText : STRING(255)
VAR_INPUT
  a : SXmlAttribute;
END_VAR
```

Sample call:

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

5.2.1.5.33 Begin

This method returns an iterator over all child elements of an XML node, always starting from the first child element. The XML node is transferred to the method as input parameter.

Syntax

```
METHOD Begin : SXmlIterator
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Sample call:

```
xmlIterator := fbXml.Begin(xmlNode);
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlNodeRef := fbXml.Node(xmlIterator);
  xmlNodeValue := fbXml.NodeText(xmlNodeRef);
  xmlIterator := fbXml.Next(xmlIterator);
END_WHILE
```

5.2.1.5.34 BeginByName

This method returns an iterator over all child elements of an XML node, starting at a particular element. The XML node is transferred to the method as input parameter.

Syntax

```
METHOD BeginByName : SXmlIterator
VAR_INPUT
  n : SXmlNode;
```

```
END_VAR
VAR_IN_OUT CONSTANT
  name : STRING;
END_VAR
```

Sample call:

```
xmlNode := fbXml.ChildByName(xmlDoc, 'Machines');
xmlIterator := fbXml.BeginByName(xmlNode, 'NameX');
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlNodeRef := fbXml.Node(xmlIterator);
  xmlNodeValue := fbXml.NodeText(xmlNodeRef);
  xmlIterator := fbXml.Next(xmlIterator);
END_WHILE
```

5.2.1.5.35 Child

This method is used to navigate through the DOM. It returns a reference to the (first) child element of the current node. The start node is transferred to the method as input parameter.

Syntax

```
METHOD ChildByName : SXmlNode
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Sample call:

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

5.2.1.5.36 ChildByAttribute

This method is used to navigate through the DOM. It returns a reference to a child element in the XML document. The start node and the name and value of the attribute are transferred to the method as input parameters.

Syntax

```
METHOD ChildByAttribute : SXmlNode
VAR_INPUT
  n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  attr : STRING;
  value : STRING;
END_VAR
```

Sample call:

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

5.2.1.5.37 ChildByAttributeAndName

This method is used to navigate through the DOM. It returns a reference to a child element in the XML document. The start node, the name and value of the attribute, and the name of the child element are transferred to the method as input parameters.

Syntax

```
METHOD ChildByAttributeAndName : SXmlNode
VAR_INPUT
  n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  attr : STRING;
  value : STRING;
  child : STRING;
END_VAR
```

Sample call:

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

5.2.1.5.38 ChildByName

This method is used to navigate through the DOM. It returns a reference to a child element in the XML document. The start node and the name of the element to be returned are transferred to the method as input parameters.

Syntax

```
METHOD ChildByName : SXmlNode
VAR_INPUT
  n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name : STRING;
END_VAR
```

Sample call:

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

5.2.1.5.39 Children

This method is used to navigate through the DOM. It returns an iterator for several child elements found in the XML document. The iterator can then be used for further navigation through the elements that were found. The start node and a reference to the iterator are transferred to the method as input parameters.

Syntax

```
METHOD Children : SXmlNode
VAR_INPUT
  n : SXmlNode;
  it : REFERENCE TO SXmlIterator;
END_VAR
```

Sample call:

```
xmlRet := fbXml.Children(xmlNode, xmlIterator);
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlMachineNodeRef := fbXml.Node(xmlIterator);
  xmlMachineNodeText := fbXml.NodeText(xmlMachineNodeRef);
  xmlIterator := fbXml.Next(xmlIterator);
END_WHILE
```

5.2.1.5.40 ChildrenByName

This method is used to navigate through the DOM. It returns an iterator for several child elements found in the XML document. The iterator can then be used for further navigation through the elements that were found. The start node, the name of the child elements to be found and a reference to the iterator are transferred to the method as input parameters.

Syntax

```
METHOD ChildrenByName : SXmlNode
VAR_INPUT
  n : SXmlNode;
  it : REFERENCE TO SXmlIterator;
END_VAR
VAR_IN_OUT CONSTANT
  name : STRING;
END_VAR
```

Sample call:

```
xmlMachineNode := fbXml.ChildrenByName(xmlMachines, xmlIterator, 'Machine');
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlMachineNodeRef := fbXml.Node(xmlIterator);
  xmlMachineNodeText := fbXml.NodeText(xmlMachineNodeRef);
  xmlIterator := fbXml.Next(xmlIterator);
END_WHILE
```

5.2.1.5.41 Compare

This method checks two iterators for equality.

Syntax

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

Sample call:

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

5.2.1.5.42 CopyAttributeText

This method reads the value of an XML attribute and writes it to a variable of data type String. The XML attribute, the target variable and the length to be written are transferred to the method as input parameters. The method returns the actual size.

Syntax

```
METHOD CopyAttributeText : UDINT
VAR_INPUT
  a : SXmlAttribute;
END_VAR
VAR_IN_OUT CONSTANT
  sXml : STRING;
END_VAR
VAR_INPUT
  nXml : UDINT;
END_VAR
```

Sample call:

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

5.2.1.5.43 CopyDocument

This method copies the contents of the DOM memory into a variable of the data type String. The length to be written and the variable into which the resulting string is to be written are transferred to the method as input parameters. The method returns the actually written length. Note that the size of the string variable is at least equal to the size of the XML document in the DOM.

Syntax

```
METHOD CopyDocument : UDINT
VAR_IN_OUT CONSTANT
  sXml : STRING;
END_VAR
VAR_INPUT
  nXml : UDINT;
END_VAR
```

Sample call:

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

5.2.1.5.44 CopyNodeText

This method reads the value of an XML node and writes it to a variable of data type String. The XML node, the target variable and the length to be written are transferred to the method as input parameters. The method returns the actual size.

Syntax

```
METHOD CopyNodeText : UDINT
VAR_INPUT
  n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  sXml : STRING;
END_VAR
VAR_INPUT
  nXml : UDINT;
END_VAR
```

Sample call:

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

5.2.1.5.45 CopyNodeXml

This method reads the XML structure of an XML node and writes it to a variable of data type String. The XML node, the target variable and the length to be written are transferred to the method as input parameters. The method returns the actual size.

Syntax

```
METHOD CopyNodeXml : UDINT
VAR_INPUT
  a : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  sXml : STRING;
END_VAR
VAR_INPUT
  nXml : UDINT;
END_VAR
```

Sample call:

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

5.2.1.5.46 FirstNodeByPath

This method navigates through an XML document using a path that was transferred to the method. The path and the start node are transferred to the method as input parameters. The path is specified with "/" as separator. The method returns a reference to the XML node that was found.

Syntax

```
METHOD FirstNodeByPath : SXmlNode
VAR_INPUT
  n : SXmlNode;
  path : STRING;
END_VAR
```

Sample call:

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

5.2.1.5.47 GetAttributeTextLength

This method returns the length of the value of an XML attribute. The XML attribute is transferred to the method as input parameter.

Syntax

```
METHOD GetAttributeTextLength : UDINT
VAR_INPUT
  a : SXmlAttribute;
END_VAR
```

Sample call:

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

5.2.1.5.48 GetDocumentLength

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

Syntax

```
METHOD GetDocumentLength : UDINT
```

Sample call:

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

5.2.1.5.49 GetDocumentNode

This method returns the root node of an XML document. This is not the same as the first XML node in the document (the method GetRootNode() should be used for this). The method can also be used to create an empty XML document in the DOM.

Syntax

```
METHOD GetDocumentNode : SXmlNode
```

Sample call:

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

5.2.1.5.50 GetNodeTextLength

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

Syntax

```
METHOD GetNodeTextLength : UDINT
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Sample call:

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

5.2.1.5.51 GetNodeXmlLength

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

Syntax

```
METHOD GetNodeXmlLength : UDINT
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Sample call:

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

5.2.1.5.52 GetRootNode

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

Syntax

```
METHOD GetRootNode : SXmlNode
```

Sample call:

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

5.2.1.5.53 InsertAttributeCopy

This method adds an attribute to an XML node. The name and value of an existing attribute are copied. The attribute can be placed at a specific position. The XML node, the position and a reference to the existing attribute object are transferred to the method as input parameters. The method returns a reference to the newly added attribute.

Syntax

```
METHOD InsertAttributeCopy : SXmlAttribute
VAR_INPUT
  n : SXmlNode;
  before : SXmlAttribute;
  copy : SXmlAttribute;
END_VAR
```

Sample call:

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

5.2.1.5.54 InsertAttribute

This method adds an attribute to an XML node. The attribute can be placed at a specific position. The XML node and the position and name of the new attribute are transferred to the method as input parameters. The method returns a reference to the newly added attribute. A value for the attribute can then be entered using the SetAttribute() method, for example.

Syntax

```
METHOD InsertAttribute : SXmlAttribute
VAR_INPUT
  n : SXmlNode;
  before : SXmlAttribute;
END_VAR
VAR_IN_OUT CONSTANT
  name : STRING;
END_VAR
```

Sample call:

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

5.2.1.5.55 InsertChild

This method adds a node to an existing XML node. The new node can be placed at a specific location. The existing XML node and the position and name of the new node are transferred to the method as input parameters. The method returns a reference to the newly added node. A value for the node can then be entered using the SetChild() method, for example.

Syntax

```
METHOD InsertChild : SXmlNode
VAR_INPUT
  n : SXmlNode;
  before : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name : STRING;
END_VAR
```

Sample call:

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

5.2.1.5.56 InsertCopy

This method adds a new node to an existing XML node and copies an existing node. The new node can be placed anywhere in the existing node. The XML node, the position and a reference to the existing node object are transferred to the method as input parameters. The method returns a reference to the newly added node.

Syntax

```
METHOD InsertCopy : SXmlNode
VAR_INPUT
  n : SXmlNode;
  before : SXmlNode;
  copy : SXmlNode;
END_VAR
```

Sample call:

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

5.2.1.5.57 IsEnd

This method checks whether a given XML iterator is at the end of the iteration that is to be performed.

Syntax

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

Sample call:

```
xmlIterator := fbXml.Begin(xmlNode);
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlNodeRef := fbXml.Node(xmlIterator);
  xmlNodeValue := fbXml.NodeText(xmlNodeRef);
  xmlIterator := fbXml.Next(xmlIterator);
END_WHILE
```

5.2.1.5.58 LoadDocumentFromFile

This method loads an XML document from a file. The absolute path to the file is transferred to the method as input parameter.

Syntax

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

Sample call:

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

5.2.1.5.59 NewDocument

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

Syntax

```
METHOD NewDocument : BOOL
```

Sample call:

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

5.2.1.5.60 Next

This method sets an XML iterator for the next object that is to be processed.

Syntax

```
METHOD Next : SXmlIterator
VAR_INPUT
  it : SXmlIterator;
END_VAR
```

Sample call:

```
xmlIterator := fbXml.Begin(xmlNode);
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlNodeRef := fbXml.Node(xmlIterator);
  xmlNodeValue := fbXml.NodeText(xmlNodeRef);
  xmlIterator := fbXml.Next(xmlIterator);
END_WHILE
```

5.2.1.5.61 NextAttribute

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

Syntax

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

Sample call:

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

5.2.1.5.62 NextByName

This method sets an XML iterator for the next object that is to be processed, which is identified by its name.

Syntax

```
METHOD NextByName : SXmlIterator
VAR_INPUT
  it : SXmlIterator;
END_VAR
VAR_IN_OUT CONSTANT
  name : STRING;
END_VAR
```

Sample call:

```
xmlIterator := fbXml.Begin(xmlNode);
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlNodeRef := fbXml.Node(xmlIterator);
  xmlNodeValue := fbXml.NodeText(xmlNodeRef);
  xmlIterator := fbXml.NextByName(xmlIterator, 'SomeName');
END_WHILE
```

5.2.1.5.63 NextSibling

This method returns the next direct node for a given XML node at the same XML level.

Syntax

```
METHOD NextSibling : SXmlNode
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Sample call:

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

5.2.1.5.64 NextSiblingByName

This method returns the next direct node for a given XML node with a particular name at the same XML level.

Syntax

```
METHOD NextSiblingByName : SXmlNode
VAR_INPUT
  n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name : STRING;
END_VAR
```

Sample call:

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

5.2.1.5.65 Node

This method is used in conjunction with an iterator to navigate through the DOM. The iterator is transferred to the method as input parameter. The method then returns the current XML node as return value.

Syntax

```
METHOD Node : SXmlNode
VAR_INPUT
  it : SXmlIterator;
END_VAR
```

Sample call:

```
xmlMachineNode := fbXml.ChildrenByName(xmlMachines, xmlIterator, 'Machine');
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlMachineNode := fbXml.Node(xmlIterator);
  xmlMachineNodeValue := fbXml.NodeText(xmlMachineNode);
  xmlIterator := fbXml.Next(xmlIterator);
END_WHILE
```

5.2.1.5.66 NodeAsBool

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

Syntax

```
METHOD NodeAsBool : BOOL
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Sample call:

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

5.2.1.5.67 NodeAsDouble

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

Syntax

```
METHOD NodeAsDouble : LREAL
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Sample call:

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

5.2.1.5.68 NodeAsFloat

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

Syntax

```
METHOD NodeAsFloat : REAL
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Sample call:

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

5.2.1.5.69 NodeAsInt

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

Syntax

```
METHOD NodeAsInt : DINT
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Sample call:

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

5.2.1.5.70 NodeAsLint

This method returns the text of an XML node as a data type Integer64. The XML node is transferred to the method as input parameter.

Syntax

```
METHOD NodeAsLint : LINT
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Sample call:

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

5.2.1.5.71 NodeAsUint

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

Syntax

```
METHOD NodeAsUint : UDINT
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Sample call:

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

5.2.1.5.72 NodeAsUlint

This method returns the text of an XML node as data type Unsigned Integer64. The XML node is transferred to the method as input parameter.

Syntax

```
METHOD NodeAsUlint : ULINT
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Sample call:

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

5.2.1.5.73 NodeName

This method returns the name of an XML node. A reference to the XML node is transferred to the method as input parameter.

Syntax

```
METHOD NodeName : STRING
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Sample call:

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

5.2.1.5.74 NodeText

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

Syntax

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

Sample call:

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

5.2.1.5.75 ParseDocument

This method loads an XML document into the DOM memory for further processing. The XML document exists as a string and is transferred to the method as input parameter. A reference to the XML document in the DOM is returned to the caller.

Syntax

```
METHOD ParseDocument : SXmlNode
VAR_IN_OUT CONSTANT
  sXml : STRING;
END_VAR
```

Sample call:

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

5.2.1.5.76 RemoveChild

This method removes an XML child node from a given XML node. The two XML nodes are transferred to the method as input parameters. The method returns TRUE if the operation was successful and the XML node was removed.

Syntax

```
METHOD RemoveChild : BOOL
VAR_INPUT
  n : SXmlNode;
  child : SXmlNode;
END_VAR
```

Sample call:

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

5.2.1.5.77 RemoveChildByName

This method removes an XML child node from a given XML node. The node to be removed is addressed by its name. If there is more than one child node, the last child node is removed. The method returns TRUE if the operation was successful and the XML node was removed.

Syntax

```
METHOD RemoveChildByName : BOOL
VAR_INPUT
  n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name : STRING;
END_VAR
```

Sample call:

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

5.2.1.5.78 SaveDocumentToFile

This method saves the current XML document in a file.

Syntax

```
METHOD SaveDocumentToFile : BOOL
VAR_IN_OUT CONSTANT
  sFile : STRING;
END_VAR
VAR_INPUT
  bExec : REFERENCE TO BOOL;
END_VAR
```

Sample call:

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

5.2.1.5.79 SetAttribute

This method sets the value of an attribute. The value has the data type String.

Syntax

```
METHOD SetAttribute : SXmlAttribute
VAR_INPUT
  a : SXmlAttribute;
END_VAR
VAR_IN_OUT CONSTANT
  value : STRING;
END_VAR
```

Sample call:

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

5.2.1.5.80 SetAttributeAsBool

This method sets the value of an attribute. The value has the data type Boolean.

Syntax

```
METHOD SetAttributeAsBool : SXmlAttribute
VAR_INPUT
  a : SXmlAttribute;
  value : BOOL;
END_VAR
```

Sample call:

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

5.2.1.5.81 SetAttributeAsDouble

This method sets the value of an attribute. The value here has the data type Double.

Syntax

```
METHOD SetAttributeAsDouble : SXmlAttribute
VAR_INPUT
  a : SXmlAttribute;
  value : LREAL;
END_VAR
```

Sample call:

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

5.2.1.5.82 SetAttributeAsFloat

This method sets the value of an attribute. The value has the data type Float.

Syntax

```
METHOD SetAttributeAsFloat : SXmlAttribute
VAR_INPUT
  a : SXmlAttribute;
  value : REAL;
END_VAR
```

Sample call:

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

5.2.1.5.83 SetAttributeAsInt

This method sets the value of an attribute. The value has the data type Integer.

Syntax

```
METHOD SetAttributeAsInt : SXmlAttribute
VAR_INPUT
  a : SXmlAttribute;
  value : DINT;
END_VAR
```

Sample call:

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

5.2.1.5.84 SetAttributeAsLint

This method sets the value of an attribute. The value has the data type Integer64.

Syntax

```
METHOD SetAttributeAsLint : SXmlAttribute
VAR_INPUT
  a : SXmlAttribute;
  value : LINT;
END_VAR
```

Sample call:

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

5.2.1.5.85 SetAttributeAsUint

This method sets the value of an attribute. The value has the data type Unsigned Integer.

Syntax

```
METHOD SetAttributeAsUint : SXmlAttribute
VAR_INPUT
  a : SXmlAttribute;
  value : UDINT;
END_VAR
```

Sample call:

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

5.2.1.5.86 SetAttributeAsUlint

This method sets the value of an attribute. The value has the data type Unsigned Integer64.

Syntax

```
METHOD SetAttributeAsUlint : SXmlAttribute
VAR_INPUT
  a : SXmlAttribute;
  value : ULINT;
END_VAR
```

Sample call:

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

5.2.1.5.87 SetChild

This method sets the value of an XML node. The value is transferred to the method as input parameter of data type String. The input parameter cdata indicates whether the value of the node is to be encapsulated in a CDATA function block, so that certain special characters such as "<" and ">" are allowed as values.

Syntax

```
METHOD SetChild : SXmlNode
VAR_INPUT
  n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  value : STRING;
END_VAR
VAR_INPUT
  cdata : BOOL;
END_VAR
```

Sample call:

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

5.2.1.5.88 SetChildAsBool

This method sets the value of an XML node. The value is transferred to the method as input parameter of data type Boolean.

Syntax

```
METHOD SetChildAsBool : SXmlNode
VAR_INPUT
  n : SXmlNode;
  value : BOOL;
END_VAR
```

Sample call:

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

5.2.1.5.89 SetChildAsDouble

This method sets the value of an XML node. The value is transferred to the method as input parameter of data type Double.

Syntax

```
METHOD SetChildAsDouble : SXmlNode
VAR_INPUT
  n : SXmlNode;
  value : LREAL;
END_VAR
```

Sample call:

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

5.2.1.5.90 SetChildAsFloat

This method sets the value of an XML node. The value is transferred to the method as input parameter of data type Float.

Syntax

```
METHOD SetChildAsFloat : SXmlNode
VAR_INPUT
  n : SXmlNode;
  value : REAL;
END_VAR
```

Sample call:

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

5.2.1.5.91 SetChildAsInt

This method sets the value of an XML node. The value is transferred to the method as input parameter of data type Integer.

Syntax

```
METHOD SetChildAsInt : SXmlNode
VAR_INPUT
  n : SXmlNode;
  value : DINT;
END_VAR
```

Sample call:

```
xmlNode := fbXml.SetChildAsInt(xmlExistingNode, 42);
```

5.2.1.5.92 SetChildAsLint

This method sets the value of an XML node. The value is transferred to the method as input parameter of data type Integer64.

Syntax

```
METHOD SetChildAsLint : SXmlNode
VAR_INPUT
  n : SXmlNode;
  value : LINT;
END_VAR
```

Sample call:

```
xmlNode := fbXml.SetChildAsLint(xmlExistingNode, 42);
```

5.2.1.5.93 SetChildAsUint

This method sets the value of an XML node. The value is transferred to the method as input parameter of data type Unsigned Integer.

Syntax

```
METHOD SetChildAsUint : SXmlNode
VAR_INPUT
  n : SXmlNode;
  value : UDINT;
END_VAR
```

Sample call:

```
xmlNode := fbXml.SetChildAsUint(xmlExistingNode, 42);
```

5.2.1.5.94 SetChildAsUlint

This method sets the value of an XML node. The value is transferred to the method as input parameter of data type Unsigned Integer64.

Syntax

```
METHOD SetChildAsUlint : SXmlNode
VAR_INPUT
  n : SXmlNode;
  value : ULINT;
END_VAR
```

Sample call:

```
xmlNode := fbXml.SetChildAsUlint(xmlExistingNode, 42);
```

5.2.2 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.

Sample	Link	Description
1	IotMqttSampleUsingQueue [▶ 106]	MQTT communication based on a messages queue
2	IotMqttSampleUsingCallback [▶ 108]	MQTT communication based on a callback method
3	IotMqttSampleTlsPsk [▶ 110]	MQTT communication via a secure TLS connection and PSK (PreSharedKey)
4	IotMqttSampleTlsCa [▶ 111]	MQTT communication via a secure TLS connection and CA (certificate authority) certificate
5	IotMqttSampleAwsIoT [▶ 112]	MQTT communication with AWS IoT
6	IotMqttSampleAzureIotHub [▶ 113]	MQTT communication with the Microsoft Azure IoT Hub
7	IotMqttSampleIbmWatsonIoT [▶ 113]	MQTT communication with IBM Watson IoT

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. Receiving of messages involves two steps. First, a general decision is made on which types of messages are to be received ("Subscribe"). Then, received messages are collected in a message queue, from where they can be read and evaluated.

Download: https://infosys.beckhoff.com/content/1033/tf6701_tc3_iot_communication_mqtt/Resources/zip/36028800560150923.zip

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 [▶ 23] and two auxiliary variables to control the program sequence, if required.

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

3. Declare two variables (for topic and payload) for the MQTT message to be sent. In the sample a message is to be sent every second.

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

4. For each message receive operation a variable containing the topic to be received should be declared, plus two further variables indicating the topic and payload of the last received message.
   The received messages are to be collected in a queue for subsequent evaluation on a one-by-one basis. To this end, you should declare an instance of FB_IotMqttMessageQueue [▶ 30] and an instance of FB_IotMqttMessage [▶ 31].

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

5. In the program part, the MQTT client must be triggered cyclically, in order to ensure that a connection to the broker is established and maintained and the message is received. Set the parameters of the desired connection and initialize the connection with the transfer parameter `bConnect := TRUE`.
   In the sample the parameters are assigned once in the program code before the client call. Since this is usually only required once, the parameters can already be specified in the declaration part during instantiation of the MQTT client. Not all parameters have to be assigned.
   In the sample the broker is local. The IP address or the name can also be specified.

```
IF bSetParameter THEN
    bSetParameter               := FALSE;
    fbMqttClient.sHostName      := 'localhost';
    fbMqttClient.nHostPort      := 1883;
//  fbMqttClient.sClientId      := 'MyTcMqttClient';
    fbMqttClient.sTopicPrefix   := '';
//  fbMqttClient.nKeepAlive     := 60;
//  fbMqttClient.sUserName      := ;
//  fbMqttClient.sUserPassword  := ;
//  fbMqttClient.stWill         := ;
//  fbMqttClient.stTLS          := ;
    fbMqttClient.ipMessageQueue := fbMessageQueue;
END_IF

fbMqttClient.Execute(bConnect);
```

6. As soon as the connection to the broker is established, the client should subscribe to a particular topic. A message should be sent every second.
   In the sample `sTopicPub = sTopicSub` applies, so that a loop-back occurs. In other applications the topics usually differ.

```
IF fbMqttClient.bConnected THEN
    IF NOT bSubscribed THEN
        bSubscribed := fbMqttClient.Subscribe(sTopic:=sTopicSub, eQoS:=TcIotMqttQos.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.

Download: https://infosys.beckhoff.com/content/1033/tf6701_tc3_iot_communication_mqtt/Resources/zip/36028800560153483.zip

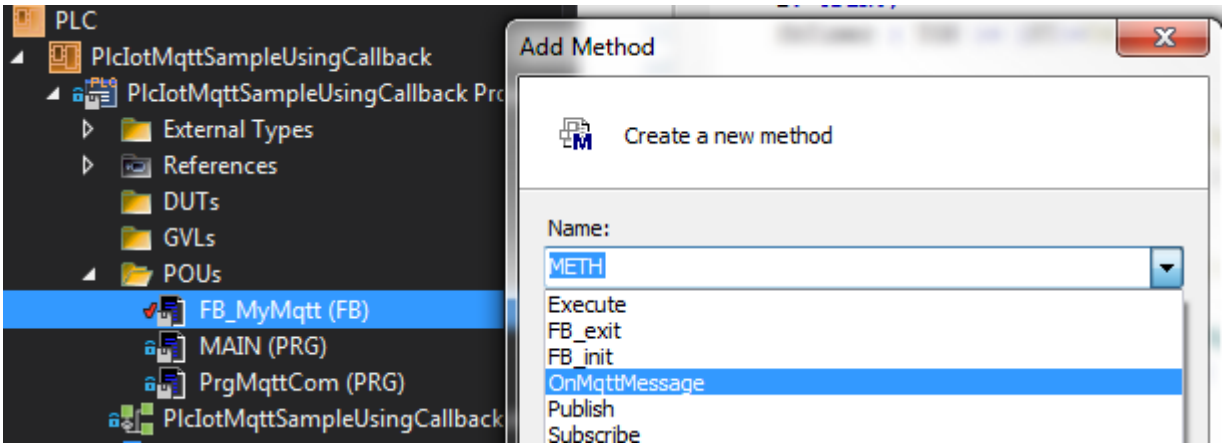
Project structure

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.

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

SUPER^.nMessagesRcv := SUPER^.nMessagesRcv + 1;

STRNCPY( ADR(sTopicRcv), ADR(topic), SIZEOF(sTopicRcv) );
MEMCPY( ADR(sPayloadRcv), payload, MIN(length, DINT_TO_UDINT(SIZEOF(sPayloadRcv))) );

OnMqttMessage := S_OK;
```

5. The other steps are similar to the sample MQTT communication via a message queue [▶ 106].
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.

Download: https://infosys.beckhoff.com/content/1033/tf6701_tc3_iot_communication_mqtt/Resources/zip/36028800559008011.zip

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 [▶ 23] 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 [▶ 106] and in particular IotMqttSampleAwsIoT [▶ 112]. 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 an AWS cloud (Amazon Web Services)

This sample illustrates the communication with the AWS IoT broker, which is part of the AWS IoT platform. The broker requires authentication via TLS client certificate. In this sample messages are sent to AWS IoT and received from it. Since this sample is essentially based on the sample IotMqttSampleUsingQueue [▶ 106], only the parts that are relevant for establishing a connection to AWS IoT are explained in this section. Also note the Instructions for using TF6701 with AWS IoT [▶ 18].

Download: https://infosys.beckhoff.com/content/1033/tf6701_tc3_iot_communication_mqtt/Resources/zip/36028800559005451.zip

Parameters for establishing a connection

The following code snippet shows the parameters required for establishing a connection to AWS 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.stTLS.sCA := 'c:\TwinCAT\3.1\Config\Certificates\root.pem';
  fbMqttClient.stTLS.sCert := 'c:\TwinCAT\3.1\Config\Certificates\7613eee18a-certificate.pem.crt';
  fbMqttClient.stTLS.sKeyFile := 'c:\TwinCAT\3.1\Config\Certificates\7613eee18a-private.pem.key';
  fbMqttClient.sHostName:= 'a35raby201xp77.iot.eu-west-1.amazonaws.com';
  fbMqttClient.nHostPort:= 8883;
  fbMqttClient.sClientId:= 'CX-12345';
  fbMqttClient.ipMessageQueue := fbMessageQueue;
END_IF
```

The following information is required for establishing a connection to AWS IoT.

Parameter	Description
fbMqttClient.stTLS.sCA	Root CA certificate, which is regarded as trusted by the AWS IoT broker. If certificates generated by the AWS IoT platform are used (“one-click certificate”), the corresponding root certificate of AWS IoT can be used. It can be downloaded from the AWS IoT website when certificates are generated.
fbMqttClient.stTLS.sCert	Certificate of the “thing”, which is required as authentication for establishing a connection to AWS IoT. If the certificates generated by the AWS IoT platform are used (“one-click certificate”), this certificate can be downloaded from the AWS IoT website, once it has been generated.
fbMqttClient.stTLS.sKeyFile	Private key of the “thing”, which is required for securing the communication link. If the certificates generated by the AWS IoT platform are used (“one-click certificate”), the private key can be downloaded from the AWS IoT website, once it has been generated.
fbMqttClient.sHostname	FQDN of the AWS IoT broker instance
fbMqttClient.nHostPort	Since a connection to the AWS IoT broker can only be established via TLS, the default MQTT TLS port is used here (8883).
fbMqttClient.sClientId	The MQTT client ID can be the same as the name of the "thing".

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 for MQTT communication with a Microsoft Azure cloud

This sample illustrates the communication with the Microsoft Azure IoT Hub, which is part of the Microsoft Azure cloud. The broker requires authentication via a so-called SAS token, which can be generated via the Azure IoT Hub platform, e.g. using the so-called Device Explorer. In this sample messages are sent to the Azure IoT Hub and received from it. Since this sample is essentially based on the sample IotMqttSampleUsingQueue [▶ 106], only the parts that are relevant for establishing a connection to the IoT Hub are explained in this section. Also note the Instructions for using TF6701 with the Azure IoT Hub [▶ 20].

Download: https://infosys.beckhoff.com/content/1033/tf6701_tc3_iot_communication_mqtt/Resources/zip/45035999814897035.zip

Parameters for establishing a connection

The following code snippet shows the parameters required for establishing a connection to the Azure IoT Hub. The parameters are essentially static parameters. These can also be specified in the declaration part during instantiation of the MQTT client.

```
IF bSetParameter THEN
  bSetParameter := FALSE;
  // fbMqttClient.stTLS.sCA := 'c:\TwinCAT\3.1\Config\Certificates\azure.crt';
  fbMqttClient.stTLS.sAzureSas := 'HostName=xxx.azure-
devices.net;DeviceId=xxx;SharedAccessSignature=SharedAccessSignature sr=xxx.azure-devices.net
%2fdevices%2fXXX&sig=121b5gJZxujK5pV%2bsFIFc2nddtpuhRuY7Tjfn8kJbtA%3d&se=1490275463';
END_IF
```

The structure element fbMqttClient.stTLS.sAzureSas enables transfer of the SAS token, which can be generated with the Azure Device Explorer. The use of the structure element fbMqttClient.stTLS.sCA is optional. By default the driver uses the path specified above in the background. A root CA that is accepted by the Azure IoT Hub should be stored under this path. The topics for publish and subscribe are specified by the Azure IoT Hub and cannot be changed.

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 IotMqttSampleIbmWatsonIoT

Sample for MQTT communication with IBM Watson IoT

This sample illustrates the communication with IBM Watson IoT, which is part of the IBM cloud. In this sample messages are sent to IBM Watson IoT and received from it. Since this sample is essentially based on the sample IotMqttSampleUsingQueue [▶ 106], only the parts that are relevant for establishing a connection are explained in this section. Also note the Instructions for using TF6701 with IBM Watson IoT [▶ 18].

Download: https://infosys.beckhoff.com/content/1033/tf6701_tc3_iot_communication_mqtt/Resources/zip/36028800646628107.zip

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.

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)

6.8 IotMqttSampleMathworksThingspeak

Communication with the MathWorks ThingSpeak Cloud is illustrated in this sample. Since this sample is essentially based on the sample IotMqttSampleUsingQueue [▶ 106], only the parts that are relevant for establishing a connection with the ThingSpeak Cloud are explained in this section.

Download: https://infosys.beckhoff.com/content/1033/tf6701_tc3_iot_communication_mqtt/Resources/zip/9007203171509899.zip

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.

```
IF bSetParameter THEN
  bSetParameter := FALSE;
  fbMqttClient.sHostName:= 'mqtt.thingspeak.com';
  fbMqttClient.nHostPort:= 1883;
  fbMqttClient.sClientId:= 'myTwinCATSystem';
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. Authentication via a password that corresponds to the MQTT API key of the ThingSpeak account can also be set at the function block via the corresponding input parameter. The user name has no purpose and can be assigned any value. For more information about the ThingSpeak CA certificate and the MQTT API key we recommend referring to the Mathworks ThingSpeak documentation.

```
IF bSetParameter THEN
  bSetParameter := FALSE;
  fbMqttClient.sHostName:= 'mqtt.thingspeak.com';
  fbMqttClient.nHostPort:= 8883;
  fbMqttClient.sClientId:= 'myTwinCATSystem';
  fbMqttClient.sUsername:= 'myUser';
  fbMqttClient.sUserPassword:= 'myUserPassword';

  stMqttTls.sVersion:= 'tlsv1.2';
  stMqttTls.sCA:= 'C:\TwinCAT\3.1\Config\Certificates\thingSpeakCa.cer';
  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 / <apiKey>

<channelID> corresponds here to the ID of the channel that was designated and configured for data reception in the MathWorks ThingSpeak portal.

<apiKey> corresponds to the access key for the channel.

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.

Data format

The MathWorks ThingSpeak Cloud currently uses its own string-based data format. This data format is generated in the function F_Mqtt_ThingSpeak_CreatePayloadStr().

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 Tc3JsonXmlSampleJsonDataType

Sample of the automatic conversion of structures into a JSON message

This sample illustrates how a data structure can be converted into a JSON message (and vice versa). In the conversion the layout of a structure is converted one-to-one into a corresponding JSON equivalent. Additional metadata can be created via PLC attributes on the member variables of the structure.

Download: https://infosys.beckhoff.com/content/1033/tf6701_tc3_iot_communication_mqtt/Resources/zip/3664376331.zip

Layout of the data structure to be converted

```
TYPE ST_Values :
STRUCT

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

  {attribute 'Unit' := 'V'}
  {attribute 'DisplayName' := 'Voltage'}
  Sensor2 : DINT;

  {attribute 'Unit' := 'A'}
  {attribute 'DisplayName' := 'Current'}
  Sensor3 : DINT;

END_STRUCT
END_TYPE
```

Declaration range

```
PROGRAM MAIN
VAR
  dtTimestamp    : DATE_AND_TIME := DT#2017-04-04-12:42:42;
  fbJson         : FB_JsonSaxWriter;
  fbJsonDataType : FB_JsonReadWriteDataType;
  sJsonDoc       : STRING(255);
  sJsonDoc2      : STRING(2000);
  stValues       : ST_Values;
END_VAR
```

Implementation range

Two ways of generating the JSON message are shown, starting with the instance fbJson of the function block FB_JsonSaxWriter. The GetDocument() method can be used with a JSON message with no more than 255 characters. However, the CopyDocument() method must be used with larger JSON messages.

```
fbJson.ResetDocument();
fbJson.StartObject();
fbjson.AddKeyDateTime('Timestamp', dtTimestamp);
fbJsonDataType.AddJsonKeyValueFromSymbol(fbJson, 'Values', 'ST_Values', SIZEOF(stValues), ADR(stValu
es));
fbJsonDataType.AddJsonKeyPropertiesFromSymbol(fbJson, 'MetaData', 'ST_Values', 'Unit|DisplayName');
fbJson.EndObject();
sJsonDoc := fbJson.GetDocument();
fbJson.CopyDocument(sJsonDoc2, SIZEOF(sJsonDoc2));
```

Resulting JSON message

```
{
  "Timestamp": "2017-04-10T17:35:49",
  "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 SetSymbol-FromJson() returns FALSE.

6.10 Tc3JsonXmlSampleJsonSaxReader

Sample of the parsing of JSON documents via SAX Reader

This sample illustrates how a JSON message can be run through programmatically. The function block FB_JsonSaxReader is used as the basis.

Download: https://infosys.beckhoff.com/content/1033/tf6701_tc3_iot_communication_mqtt/Resources/zip/3664750475.zip

Declaration range

```
PROGRAM MAIN
VAR
  fbJson     : FB_JsonSaxReader;
  pJsonParse : JsonSaxHandler;
  sJsonDoc   : STRING(255) := '{"Values":
{"Timestamp":"2017-04-04T12:42:42","Sensor1":42.42,"Sensor2":42}}';
END_VAR
```

Implementation range

Through the calling of the Parse() method, the transfer of the JSON message as a STRING and the interface pointer to a function block instance that implements the interface ItcJsonSaxHandler, the SAX Reader is activated and the corresponding callback methods are run through.

```
fbJson.Parse(sJson := sJsonDoc, ipHdl := pJsonParse);
```

Callback methods

The callback methods are called on the instance of the function block that implements the interface ItcJsonSaxHandler. Each callback method represents a "found" element in the JSON message. For example, the callback method OnStartObject() is called as soon as an opening curly bracket has been detected. According to the example JSON message mentioned above, therefore, the following callback methods are run through in this order:

1. OnStartObject(), due to the first opening curly bracket
2. OnKey(), due to the property "Values"
3. OnStartObject(), due to the second opening curly bracket
4. OnKey(), due to the property "Timestamp"
5. OnString(), due to the value of the property "Timestamp"
6. OnKey(), due to the property "Sensor1"
7. OnLreal(), due to the value of the property "Sensor1"
8. OnKey(), due to the property "Sensor2"
9. OnUdint(), due to the value of the property "Sensor2"
10. OnEndObject(), due to the first closing curly bracket

11. OnEndObject(), due to the second closing curly bracket

Within the callback methods the current state is defined and saved via an instance of the enum E_JsonStates. This can also be used to determine whether the JSON message is valid. For example, if the callback method OnLreal() is called and the state is not the expected State 70 (JSON_STATE_ONLREAL), the return value S_FALSE can be returned to the method. The SAX Reader then automatically cancels the further processing.

6.11 Tc3JsonXmlSampleJsonSaxWriter

Sample of the creation of JSON documents via SAX Writer

This sample illustrates how a JSON message can be created over the DAX mechanism. The function block FB_JsonSaxWriter is used as the basis.

Download: https://infosys.beckhoff.com/content/1033/tf6701_tc3_iot_communication_mqtt/Resources/zip/3664753419.zip

Declaration range

```
PROGRAM MAIN
VAR
  dtTimestamp : DATE_AND_TIME := DT#2017-04-04-12:42:42;
  fbJson      : FB_JsonSaxWriter;
  sJsonDoc    : STRING(255);
END_VAR
```

Implementation range

The SAX mechanism runs sequentially through the JSON document to be created, i.e. the corresponding elements are run 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
  }
}
```

6.12 Tc3JsonXmlSampleJsonDomReader

This sample illustrates how a JSON message can be run through programmatically on the basis of DOM. The function block FB_JsonDomParser is used as the basis.

Download: https://infosys.beckhoff.com/content/1033/tf6701_tc3_iot_communication_mqtt/Resources/zip/3916597387.zip

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
```

6.13 Tc3JsonXmlSampleXmlDomReader

This sample illustrates how an XML document can be processed programmatically based on DOM. The function block FB_XmlDomParser is used as a basis.

Download: https://infosys.beckhoff.com/content/1033/tf6701_tc3_iot_communication_mqtt/Resources/zip/5529225227.zip

Declaration range

```
PROGRAM MAIN
VAR
  fbXml : FB_XmlDomParser;
  xmlDoc : SXmlNode;
  xmlMachines : SXmlNode;
  xmlMachine1 : SXmlNode;
  xmlMachine2 : SXmlNode;
  xmlIterator : SXmlIterator;
  xmlMachineNode : SXmlNode;
  xmlMachineNodeValue : STRING;
  xmlMachineAttributeRef : SXmlAttribute;
  xmlMachine1Attribute : SXmlAttribute;
  xmlMachine2Attribute : SXmlAttribute;
  sMachine1Name : STRING;
  sMachine2Name : STRING;
  nMachineAttribute : DINT;
  nMachine1Attribute : DINT;
  nMachine2Attribute : DINT;
  sMessageToParse : STRING(255) := '<Machines><Machine Type="1" Test="3">Wilde Nelli</
Machine><Machine Type="2">Huber8</Machine></Machines>';
END_VAR
```

Implementation range

The implementation section shows various options for parsing an XML document.

```
(* Load XML content *)
xmlDoc := fbXml.ParseDocument(sMessageToParse);

(* Parse XML nodes - Option 1 *)
xmlMachines := fbXml.ChildByName(xmlDoc, 'Machines');
xmlMachine1 := fbXml.ChildByAttribute(xmlMachines, 'Type', '1');
xmlMachine2 := fbXml.ChildByAttributeAndName(xmlMachines, 'Type', '2', 'Machine');

(* Parse XML nodes - Option 2 *)
xmlMachines := fbXml.ChildByName(xmlDoc, 'Machines');
xmlMachineNode := fbXml.Children(xmlMachines, xmlIterator);
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlMachineNode := fbXml.Node(xmlIterator);
  xmlMachineNodeValue := fbXml.NodeText(xmlMachineNode);
  xmlMachineNode := fbXml.Children(xmlMachines, xmlIterator);
END_WHILE

(* Parse XML nodes - Option 3 *)
xmlMachines := fbXml.ChildByName(xmlDoc, 'Machines');
xmlIterator := fbXml.Begin(xmlMachines);
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlMachineNode := fbXml.Node(xmlIterator);
  xmlMachineNodeValue := fbXml.NodeText(xmlMachineNode);
  xmlIterator := fbXml.Next(xmlIterator);
  xmlIterator := fbXml.End(xmlMachines);
END_WHILE

(* Parse XML attributes - Option 1*)
xmlMachine1Attribute := fbXml.Attribute(xmlMachine1, 'Type');
xmlMachine2Attribute := fbXml.Attribute(xmlMachine2, 'Type');

(* Parse XML attributes - Option 2*)
xmlIterator := fbXml.AttributeBegin(xmlMachine1);
WHILE NOT fbXml.IsEnd(xmlIterator) DO
```

```
  xmlMachineAttributeRef := fbXml.AttributeFromIterator(xmlIterator);
  nMachineAttribute := fbXml.AttributeAsInt(xmlMachineAttributeRef);
  xmlIterator := fbXml.Next(xmlIterator);
END_WHILE

(* Retrieve node values *)
sMachine1Name := fbXml.NodeText(xmlMachine1);
sMachine2Name := fbXml.NodeText(xmlMachine2);

(* Retrieve attribute values *)
nMachine1Attribute := fbXml.AttributeAsInt(xmlMachine1Attribute);
nMachine2Attribute := fbXml.AttributeAsInt(xmlMachine2Attribute);
```

6.14 Tc3JsonXmlSampleXmlDomWriter

This sample illustrates how an XML document can be created programmatically based on DOM. The function block FB_XmlDomParser is used as a basis.

Download: https://infosys.beckhoff.com/content/1033/tf6701_tc3_iot_communication_mqtt/Resources/zip/5529228299.zip

Declaration range

```
PROGRAM MAIN
VAR
  fbXml : FB_XmlDomParser;
  objRoot : SXmlNode;
  objMachines : SXmlNode;
  objMachine : SXmlNode;
  objControllers : SXmlNode;
  objController : SXmlNode;
  objAttribute : SXmlAttribute;
  sXmlString : STRING(1000);
  bCreate : BOOL := FALSE;
  bSave : BOOL := TRUE;
  nLength : UDINT;
  newAttr : SXmlAttribute;
END_VAR
```

Implementation range

The implementation section shows various options for creating an XML document.

```
IF bCreate THEN
  (* Create an empty XML document *)
  objRoot := fbXml.GetDocumentNode();

  (* Create a new XML node 'Machines' and add to the empty document *)
  objMachines := fbXml.AppendNode(objRoot, 'Machines');

  (* Create a new XML node 'Machine' and add an attribute to this node. Append node to 'Machines' *)
  objMachine := fbXml.AppendNode(objMachines, 'Machine');
  objAttribute := fbXml.AppendAttribute(objMachine, 'Name', 'Wilde Nelli');

  (* Create a new XML node 'Controllers' and add to the 'Machine' node *)
  objControllers := fbXml.AppendNode(objMachine, 'Controllers');

  (* Create a new XML node 'Controller' and add some attributes. Append node to 'Controllers'. *)
  objController := fbXml.AppendChild(objControllers, 'Controller', 'CX5120', FALSE);
  objAttribute := fbXml.AppendAttribute(objController, 'Type', 'EPC');
  objAttribute := fbXml.AppendAttribute(objController, 'OS', 'Windows Embedded Compact 7');

  (* Create a new XML node 'Controller' and add some attributes. Append node to 'Controllers'. *)
  objController := fbXml.AppendChild(objControllers, 'Controller', 'CX2040', FALSE);
  objAttribute := fbXml.AppendAttribute(objController, 'Type', 'EPC');
  objAttribute := fbXml.AppendAttribute(objController, 'OS', 'Windows Embedded Standard 7');

  (* Create a new XML node 'Controller' and add some attributes. Append node to 'Controllers'. *)
  objController := fbXml.AppendChild(objControllers, 'Controller', 'C6015', FALSE);
  objAttribute := fbXml.AppendAttribute(objController, 'Type', 'IPC');
  objAttribute := fbXml.AppendAttribute(objController, 'OS', 'Windows 10 IoT Enterprise');

  (* Create a new XML node 'Machine' and add an attribute to this node. Append node to 'Machines' *)
  objMachine := fbXml.AppendNode(objMachines, 'Machine');
  objAttribute := fbXml.AppendAttribute(objMachine, 'Name', 'Stanze Oscar');
```

```
  (* Create a new XML node 'Controllers' and add to the 'Machine' node *)
  objControllers := fbXml.AppendNode(objMachine, 'Controllers');

  (* Create a new XML node 'Controller' and add some attributes. Append node to 'Controllers'. *)
  objController := fbXml.AppendChild(objControllers, 'Controller', 'C6017', FALSE);
  objAttribute := fbXml.AppendAttribute(objController, 'Type', 'IPC');
  objAttribute := fbXml.AppendAttribute(objController, 'OS', 'Windows 10 IoT Enterprise');
  newAttr := fbXml.InsertAttribute(objController, objAttribute, 'AddAttribute');
  fbXml.SetAttribute(newAttr, 'Hola');

  (* Retrieve XML document and store in a variable of data type STRING(1000) *)
  nLength := fbXml.CopyDocument(sXmlString, SIZEOF(sXmlString));
  bCreate := FALSE;
END_IF
```

7 Appendix

7.1 Error Codes

In the event of an error, the function block FB_IotMqttClient [▶ 23] sets the output bError and indicates the error with hrErrorCode. All errors are listed in section “ADS Return Codes [▶ 123]”.

In addition, the output eConnectionState indicates the state of the connection between the client and the MQTT broker at all times. The enumeration offers the following possible states:

```
TYPE ETcIotMqttClientState :
(
   MQTT_ERR_CONN_PENDING:=-1,
   MQTT_ERR_SUCCESS:=0,
   MQTT_ERR_NOMEM:=1,
   MQTT_ERR_PROTOCOL:=2,
   MQTT_ERR_INVAL:=3,
   MQTT_ERR_NO_CONN:=4,
   MQTT_ERR_CONN_REFUSED:=5,
   MQTT_ERR_NOT_FOUND:=6,
   MQTT_ERR_CONN_LOST:=7,
   MQTT_ERR_TLS:=8,
   MQTT_ERR_PAYLOAD_SIZE:=9,
   MQTT_ERR_NOT_SUPPORTED:=10,
   MQTT_ERR_AUTH:=11,
   MQTT_ERR_ACL_DENIED:=12,
   MQTT_ERR_UNKNOWN:=13,
   MQTT_ERR_ERRNO:=14,
   MQTT_ERR_EAI:=15,
   MQTT_ERR_PROXY:=16
) DINT;
END_TYPE
```

7.2 ADS Return Codes

Error codes: 0x000 [▶ 123]..., 0x500 [▶ 124]..., 0x700 [▶ 124]..., 0x1000 [▶ 125]...

Global Error Codes

Hex	Dec	Description
0x0	0	no error
0x1	1	Internal error
0x2	2	No Rtime
0x3	3	Allocation locked memory error
0x4	4	Insert mailbox error
0x5	5	Wrong receive HMSG
0x6	6	target port not found
0x7	7	target machine not found
0x8	8	Unknown command ID
0x9	9	Bad task ID
0xA	10	No IO
0xB	11	Unknown ADS command
0xC	12	Win 32 error
0xD	13	Port not connected
0xE	14	Invalid ADS length
0xF	15	Invalid AMS Net ID
0x10	16	Low Installation level
0x11	17	No debug available
0x12	18	Port disabled
0x13	19	Port already connected
0x14	20	ADS Sync Win32 error
0x15	21	ADS Sync Timeout

Hex	Dec	Description
0x16	22	ADS Sync AMS error
0x17	23	ADS Sync no index map
0x18	24	Invalid ADS port
0x19	25	No memory
0x1A	26	TCP send error
0x1B	27	Host unreachable
0x1C	28	Invalid AMS fragment

Router Error Codes

Hex	Dec	Name	Description
0x500	1280	ROUTERERR_NOLOCKEDMEMORY	No locked memory can be allocated
0x501	1281	ROUTERERR_RESIZEMEMORY	The size of the router memory could not be changed
0x502	1282	ROUTERERR_MAILBOXFULL	The mailbox has reached the maximum number of possible messages. The current sent message was rejected
0x503	1283	ROUTERERR_DEBUGBOXFULL	The mailbox has reached the maximum number of possible messages. The sent message will not be displayed in the debug monitor
0x504	1284	ROUTERERR_UNKNOWNPORTTYPE	Unknown port type
0x505	1285	ROUTERERR_NOTINITIALIZED	Router is not initialized
0x506	1286	ROUTERERR_PORTALREADYINUSE	The desired port number is already assigned
0x507	1287	ROUTERERR_NOTREGISTERED	Port not registered
0x508	1288	ROUTERERR_NOMOREQUEUES	The maximum number of Ports reached
0x509	1289	ROUTERERR_INVALIDPORT	Invalid port
0x50A	1290	ROUTERERR_NOTACTIVATED	TwinCAT Router not active

General ADS Error Codes

Hex	Dec	Name	Description
0x700	1792	ADSERR_DEVICE_ERROR	error class <device error>
0x701	1793	ADSERR_DEVICE_SRVNOTSUPP	Service is not supported by server
0x702	1794	ADSERR_DEVICE_INVALIDGRP	invalid index group
0x703	1795	ADSERR_DEVICE_INVALIDOFFSET	invalid index offset
0x704	1796	ADSERR_DEVICE_INVALIDACCESS	reading/writing not permitted
0x705	1797	ADSERR_DEVICE_INVALIDSIZE	parameter size not correct
0x706	1798	ADSERR_DEVICE_INVALIDDATA	invalid parameter value(s)
0x707	1799	ADSERR_DEVICE_NOTREADY	device is not in a ready state
0x708	1800	ADSERR_DEVICE_BUSY	device is busy
0x709	1801	ADSERR_DEVICE_INVALIDCONTEXT	invalid context (must be in Windows)
0x70A	1802	ADSERR_DEVICE_NOMEMORY	out of memory
0x70B	1803	ADSERR_DEVICE_INVALIDPARM	invalid parameter value(s)
0x70C	1804	ADSERR_DEVICE_NOTFOUND	not found (files, ...)
0x70D	1805	ADSERR_DEVICE_SYNTAX	syntax error in command or file
0x70E	1806	ADSERR_DEVICE_INCOMPATIBLE	objects do not match
0x70F	1807	ADSERR_DEVICE_EXISTS	object already exists
0x710	1808	ADSERR_DEVICE_SYMBOLNOTFOUND	symbol not found
0x711	1809	ADSERR_DEVICE_SYMBOLVERSIONINVAL	symbol version invalid
0x712	1810	ADSERR_DEVICE_INVALIDSTATE	server is in invalid state
0x713	1811	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransMode not supported
0x714	1812	ADSERR_DEVICE_NOTIFYHNDINVALID	Notification handle is invalid
0x715	1813	ADSERR_DEVICE_CLIENTUNKNOWN	Notification client not registered
0x716	1814	ADSERR_DEVICE_NOMOREHDLS	no more notification handles
0x717	1815	ADSERR_DEVICE_INVALIDWATCHSIZE	size for watch too big
0x718	1816	ADSERR_DEVICE_NOTINIT	device not initialized
0x719	1817	ADSERR_DEVICE_TIMEOUT	device has a timeout
0x71A	1818	ADSERR_DEVICE_NOINTERFACE	query interface failed
0x71B	1819	ADSERR_DEVICE_INVALIDINTERFACE	wrong interface required

Hex	Dec	Name	Description
0x71C	1820	ADSERR_DEVICE_INVALIDCLSID	class ID is invalid
0x71D	1821	ADSERR_DEVICE_INVALIDOBJID	object ID is invalid
0x71E	1822	ADSERR_DEVICE_PENDING	request is pending
0x71F	1823	ADSERR_DEVICE_ABORTED	request is aborted
0x720	1824	ADSERR_DEVICE_WARNING	signal warning
0x721	1825	ADSERR_DEVICE_INVALIDARRAYIDX	invalid array index
0x722	1826	ADSERR_DEVICE_SYMBOLNOTACTIVE	symbol not active
0x723	1827	ADSERR_DEVICE_ACCESSDENIED	access denied
0x724	1828	ADSERR_DEVICE_LICENSENOTFOUND	missing license
0x725	1829	ADSERR_DEVICE_LICENSEEXPIRED	license expired
0x726	1830	ADSERR_DEVICE_LICENSEEXCEEDED	license exceeded
0x727	1831	ADSERR_DEVICE_LICENSEINVALID	license invalid
0x728	1832	ADSERR_DEVICE_LICENSESYSTEMID	license invalid system id
0x729	1833	ADSERR_DEVICE_LICENSENOTIMELIMIT	license not time limited
0x72A	1834	ADSERR_DEVICE_LICENSEFUTUREISSUE	license issue time in the future
0x72B	1835	ADSERR_DEVICE_LICENSETIMETOLONG	license time period to long
0x72c	1836	ADSERR_DEVICE_EXCEPTION	exception occured during system start
0x72D	1837	ADSERR_DEVICE_LICENSEDUPLICATED	License file read twice
0x72E	1838	ADSERR_DEVICE_SIGNATUREINVALID	invalid signature
0x72F	1839	ADSERR_DEVICE_CERTIFICATEINVALID	public key certificate
0x740	1856	ADSERR_CLIENT_ERROR	Error class <client error>
0x741	1857	ADSERR_CLIENT_INVALIDPARM	invalid parameter at service
0x742	1858	ADSERR_CLIENT_LISTEMPTY	polling list is empty
0x743	1859	ADSERR_CLIENT_VARUSED	var connection already in use
0x744	1860	ADSERR_CLIENT_DUPLINVOKEID	invoke ID in use
0x745	1861	ADSERR_CLIENT_SYNCTIMEOUT	timeout elapsed
0x746	1862	ADSERR_CLIENT_W32ERROR	error in win32 subsystem
0x747	1863	ADSERR_CLIENT_TIMEOUTINVALID	Invalid client timeout value
0x748	1864	ADSERR_CLIENT_PORTNOTOPEN	ads-port not opened
0x750	1872	ADSERR_CLIENT_NOAMSADDR	internal error in ads sync
0x751	1873	ADSERR_CLIENT_SYNCINTERNAL	hash table overflow
0x752	1874	ADSERR_CLIENT_ADDHASH	key not found in hash
0x753	1875	ADSERR_CLIENT_REMOVEHASH	no more symbols in cache
0x754	1876	ADSERR_CLIENT_NOMORESYM	invalid response received
0x755	1877	ADSERR_CLIENT_SYNCRESINVALID	sync port is locked

RTime Error Codes

Hex	Dec	Name	Description
0x1000	4096	RTERR_INTERNAL	Internal fatal error in the TwinCAT real-time system
0x1001	4097	RTERR_BADTIMERPERIODS	Timer value not vaild
0x1002	4098	RTERR_INVALIDTASKPTR	Task pointer has the invalid value ZERO
0x1003	4099	RTERR_INVALIDSTACKPTR	Task stack pointer has the invalid value ZERO
0x1004	4100	RTERR_PRIOEXISTS	The demand task priority is already assigned
0x1005	4101	RTERR_NOMORETCB	No more free TCB (Task Control Block) available. Maximum number of TCBs is 64
0x1006	4102	RTERR_NOMORESEMAS	No more free semaphores available. Maximum number of semaphores is 64
0x1007	4103	RTERR_NOMOREQUEUES	No more free queue available. Maximum number of queue is 64
0x100D	4109	RTERR_EXTIRQALREADYDEF	An external synchronization interrupt is already applied
0x100E	4110	RTERR_EXTIRQNOTDEF	No external synchronization interrupt applied
0x100F	4111	RTERR_EXTIRQINSTALLFAILED	The apply of the external synchronization interrupt failed
0x1010	4112	RTERR_IRQLNOTLESSOREQUAL	Call of a service function in the wrong context
0x1017	4119	RTERR_VMXNOTSUPPORTED	Intel VT-x extension is not supported
0x1018	4120	RTERR_VMXDISABLED	Intel VT-x extension is not enabled in system BIOS
0x1019	4121	RTERR_VMXCONTROLSMISSING	Missing function in Intel VT-x extension
0x101A	4122	RTERR_VMXENABLEFAILS	Enabling Intel VT-x fails

TCP Winsock Error Codes

Hex	Dec	Description
0x274d	10061	A connection attempt failed because the connected party did not properly respond after a period of time, or established connection failed because connected host has failed to respond.
0x2751	10065	No connection could be made because the target machine actively refused it. This error normally occurs when you try to connect to a service which is inactive on a different host - a service without a server application.
0x274c	10060	No route to a host. A socket operation was attempted to an unreachable host
		Further Winsock error codes: Win32 Error Codes

7.3 Troubleshooting

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 [▶ 23].
Connection to AWS IoT [▶ 18] 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 [▶ 23].

Behavior	Category	Description
Connection to Azure IoT Hub [▶ 20] 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 [▶ 23].
Messages do not arrive at the Azure IoT Hub [▶ 20]	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 [▶ 20] or the MSDN documentation.
Messages do not arrive at IBM Watson IoT [▶ 18]	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 [▶ 18] 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.

7.4 Support and Service

Beckhoff and their partners around the world offer comprehensive support and service, making available fast and competent assistance with all questions related to Beckhoff products and system solutions.

Beckhoff's branch offices and representatives

Please contact your Beckhoff branch office or representative for local support and service on Beckhoff products!

The addresses of Beckhoff's branch offices and representatives round the world can be found on her internet pages:
http://www.beckhoff.com

You will also find further documentation for Beckhoff components there.

Beckhoff Headquarters

Beckhoff Automation GmbH & Co. KG

Huelshorstweg 20
33415 Verl
Germany

Phone:	+49(0)5246/963-0
Fax:	+49(0)5246/963-198
e-mail:	info@beckhoff.com

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(0)5246/963-157
Fax:	+49(0)5246/963-9157
e-mail:	support@beckhoff.com

Beckhoff Service

The Beckhoff Service Center supports you in all matters of after-sales service:

- on-site service
- repair service
- spare parts service
- hotline service

Hotline:	+49(0)5246/963-460
Fax:	+49(0)5246/963-479
e-mail:	service@beckhoff.com