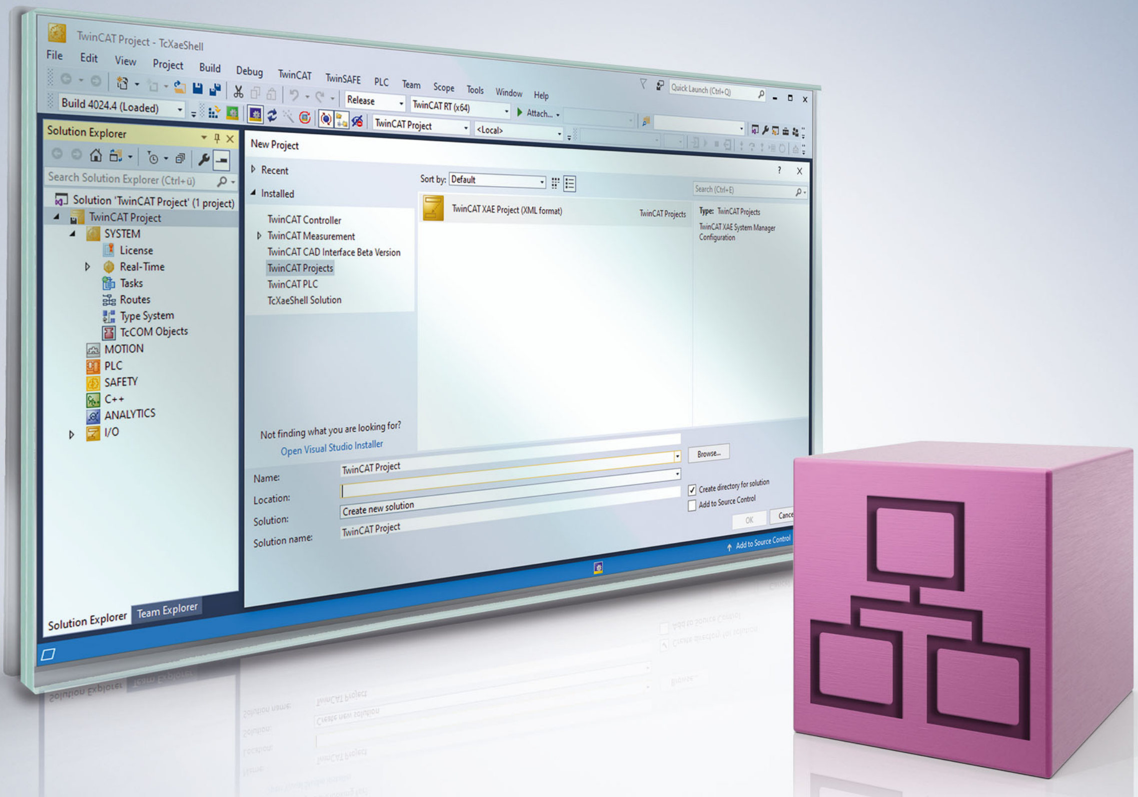
BECKHOFF New Automation Technology

Manual | EN

TF6760

TwinCAT 3 | IoT HTTPS/REST



19.09.2022 | Version: 1.9

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 applicable national standards.
It is essential that the documentation and the following notes and explanations are followed when installing and commissioning the components.
It is the duty of the technical personnel to use the documentation published at the respective time of each installation and commissioning.

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

Disclaimer

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

Trademarks

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® and XPlanar® are registered trademarks of and licensed by Beckhoff Automation GmbH.
Other designations used in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owners.

Patent Pending

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

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

Copyright

© Beckhoff Automation GmbH & Co. KG, Germany.
The reproduction, distribution and utilization of this document as well as the communication of its contents to others without express authorization are prohibited.
Offenders will be held liable for the payment of damages. All rights reserved in the event of the grant of a patent, utility model or design.

1.2 Safety instructions

Safety regulations

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

Exclusion of liability

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

Personnel qualification

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

Description of symbols

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

⚠ DANGER

Serious risk of injury!

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

⚠ WARNING

Risk of injury!

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

⚠ CAUTION

Personal injuries!

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

NOTE

Damage to the environment or devices

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

ℹ **Tip or pointer**

This symbol indicates information that contributes to better understanding.

1.3 Notes on information security

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

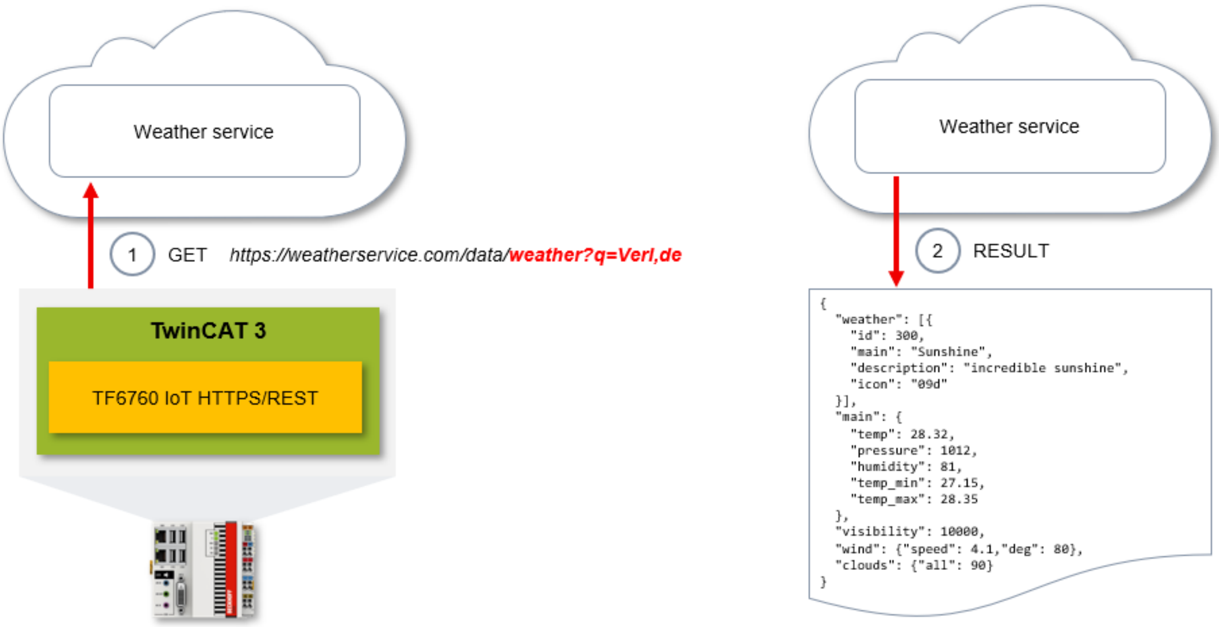
In addition, the recommendations from Beckhoff regarding appropriate protective measures should be observed. Further information regarding information security and industrial security can be found in our https://www.beckhoff.com/secguide.

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

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

2 Overview

The function blocks of the PLC library Tc3_IoTBase can be used to address REST APIs as a client via HTTP/HTTPS communication. For that common HTTP commands such as GET, PUT or POST are provided. Basically, data can be requested from a REST API and can also be sent to a REST API.



```
{
  "weather": [{
    "id": 300,
    "main": "Sunshine",
    "description": "incredible sunshine",
    "icon": "09d"
  }],
  "main": {
    "temp": 28.32,
    "pressure": 1012,
    "humidity": 81,
    "temp_min": 27.15,
    "temp_max": 28.35
  },
  "visibility": 10000,
  "wind": {"speed": 4.1,"deg": 80},
  "clouds": {"all": 90}
}
```

Fig. 1: TF6760

Product components

The function TF6760 IoT HTTPS/REST consists of the following components, which are supplied with TwinCAT 3 as standard:

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

3 Installation

3.1 System requirements

Technical data	Description
Operating system	Windows 7/10, Windows Embedded Standard 7, Windows CE 7
Target platform	PC architecture (x86, x64 or ARM)
TwinCAT version	TwinCAT 3.1 Build 4024.7 or higher
Required TwinCAT setup level	TwinCAT 3 XAE, XAR
Required TwinCAT license	TF6760 TC3 IoT HTTPS/REST

3.2 Installation

A seperate setup is not required for TF6760 IoT HTTPS/REST. 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).

Licensing the full version of a TwinCAT 3 Function

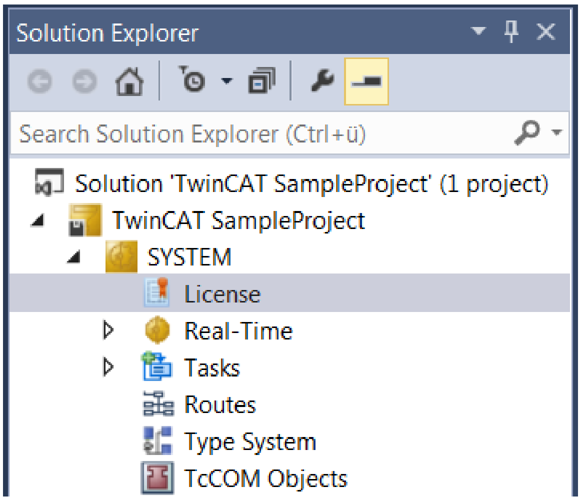
A description of the procedure to license a full version can be found in the Beckhoff Information System in the documentation "TwinCAT 3 Licensing".

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

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

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

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



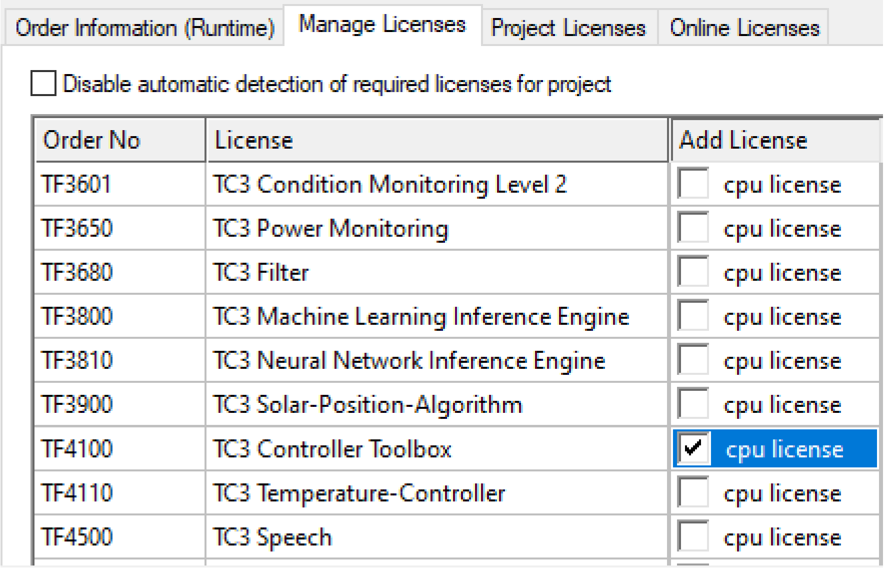
Solution Explorer

Search Solution Explorer (Ctrl+ü)

- Solution 'TwinCAT SampleProject' (1 project)
  - TwinCAT SampleProject
    - SYSTEM
      - License
      - Real-Time
      - Tasks
      - Routes
      - Type System
      - TcCOM Objects

⇨ The TwinCAT 3 license manager opens.

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



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

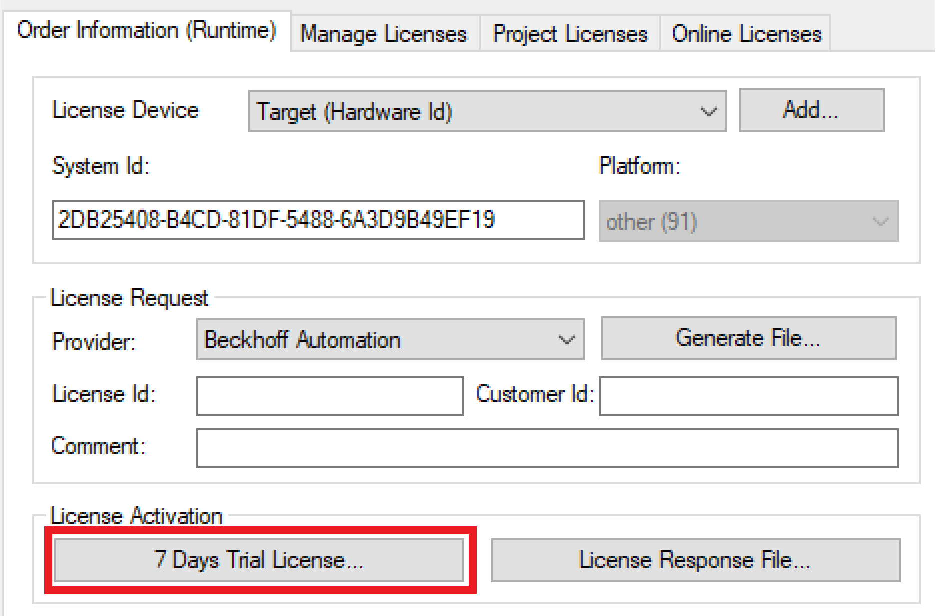
□ Disable automatic detection of required licenses for project

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

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

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

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



License Device: Target (Hardware Id)

System Id: 2DB25408-B4CD-81DF-5488-6A3D9B49EF19

Platform: other (91)

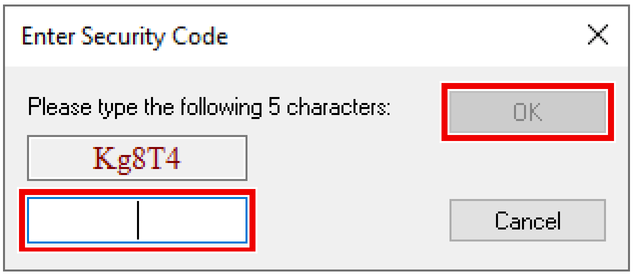
Provider: Beckhoff Automation

License Id:

Customer Id:

Comment:

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



Enter Security Code

Please type the following 5 characters:

Kg8T4

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

⇨ The 7-day trial version is enabled.

4 Technical introduction

The following chapter gives a brief technical introduction about the HTTP protocol and its secure extension, the HTTPS protocol. Furthermore the REST architecture and its connection to the HTTP protocol are described.

4.1 HTTP/HTTPS

HTTP (Hypertext Transfer Protocol) is a standard protocol which is mainly used for the IP-communication between servers and clients in the internet. The client is usually a web browser which is requesting a website from a webserver. Another use case for the HTTP protocol are REST-Webservices (Representational State Transfer).

Furthermore, is HTTP a stateless protocol, every command is handled independently. After a server has responded to a client request the connection is closed again. A client has the possibility to tell the server that the connection should be kept alive after a request.

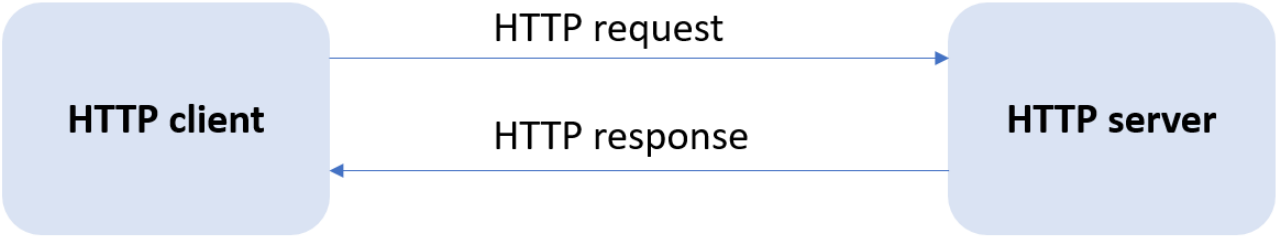


Fig. 2: HTTP communication

HTTP methods

The HTTP standard (version 1.1) defines different methods that a HTTP server can offer to requesting clients. Especially GET, POST and PUT are some of the most frequent used HTTP methods. The following table contains all HTTP methods that are defined in the standard.

HTTP method	Description
GET	Requests a resource from the server.
POST	Transfers data to the server.
PUT	Replaces or creates a resource on the server with the request payload.
CONNECT	Establishes a SSL tunnel to a server.
DELETE	Deletes the addressed resource from the server.
HEAD	Same functionality as GET, without payload data.
OPTIONS	Requests the available communication options from the server.
TRACE	Performs a message loopback for testing purposes.
PATCH	Same functionality as PUT, but used for partial changes of resources.

It is important to know for a user, that not every server implements all of these methods. According to the specification only GET and HEAD are mandatory for a server that is conform to the specification, all other commands are optional. There is also the possibility that user credentials are required to access resources on a server.

HTTP status codes

A HTTP response is divided into header and body. An important part of this response is the HTTP status code. The HTTP status code delivers information about the request back to the client. The following table shows the defined ranges of the HTTP status code.

Status code range	Description
100-199	Status codes that contain information messages.
200-299	Status codes that inform the client about a successful request.
300-399	Status codes that show a diversion and request an interaction from the user.
400-499	Status codes that show error messages (triggered by the client).
500-599	Status codes that show error messages (triggered by the server).

HTTPS

HTTPS (Hypertext Transfer Protocol Secure) is an extension of the HTTP protocol, which implements TLS (Transport Layer Security). Data sent with the HTTP protocol is totally unencrypted and does not meet the security requirements for sending e. g. passwords or credit card information. Additional to the encryption of the transported data, HTTPS also delivers server authentication.

4.2 REST API

A RESTful (Representational State Transfer) web service is a web service that is designed with an architecture specially for distributed systems. All RESTful web services on a server together are called REST API (Application Programming Interface).

REST APIs are often associated with the HTTP protocol but are neither a standard nor a protocol. So, there are – per definition - different protocols that can be used to communicate with REST APIs, HTTP is only the most common one.

A REST API has to follow six different architectural definitions that are mentioned in the specification:

- **Client-Server model:** Data and user interface have to be divided by definition.
- **Statelessness:** Client and server have to communicate stateless. Every client request must deliver all the information that the server needs to process the request.
- **Caching:** Information can be classified as cacheable or not-cacheable. A client can cache server responses for future requests, but this information might be outdated.
- **Consistent interface:** REST interfaces must be usable from different endpoints in the same way, due to their consistent interface.
- **Layered systems:** A REST API has to be a multilayered, hierarchical built system.
- **Code-On-Demand (optional):** The functions of clients have to be extendable with reloadable and executable program parts.

How to request resources from a REST API

As mentioned before, HTTP(S) is just one of the possibilities to implement a REST API but is the only one that has importance for this documentation. The following section will contain a short description how to request resources from a REST API that is implemented with HTTP(S).

A resource on a HTTPS REST API is identified with its URL. There are two different possibilities how a URL can be offered by a webservice. On the one hand, a URL can only consist of path parameters and on the other hand it could contain query parameters. The following example will explain the difference between these two possibilities.

A URL is built with three parts:

1. Domain name
2. Path parameters
3. Query parameters

When the weather for the city Verl should be requested from a fictional Beckhoff weather service, the URL without query parameters could look like:

https://www.beckhoff-weather.com/forecast/city/verl

With the usage of query parameters, the URL could look like:

https://www.beckhoff-weather.com/forecast?city=verl

The first part **https://www.beckhoff-weather.com** is the domain name of the webserver, which is translated to an IP address with the help of a DNS-Server. The path parameters **/forecast/city/verl** specify on which resource of the server the request should be processed. Another possibility is to use a query parameter like **forecast?city=verl** to request a specific resource on a path.

4.3 Technical restrictions

This chapter describes technical limitations of the IoT driver related to HTTP client functionality. It should be noted that these limitations are not flaws but design features that have been implemented deliberately.

Restriction	Description
Number of simultaneous HTTP requests from an HTTP client instance	Each instance of the function block FB_IotHttpClient [▶ 22] can only send one HTTP request at a time. The next request can only be sent once the corresponding HTTP response has arrived. If several requests are sent from the same client instance before the response arrives, they are placed in a queue. The timeout configured for the HTTP client instance has higher priority than the response.
Communication of large files	When communicating large files, the user should be aware that this will cause considerable real-time load, since a large amount of data has to be copied. Router memory is also used.

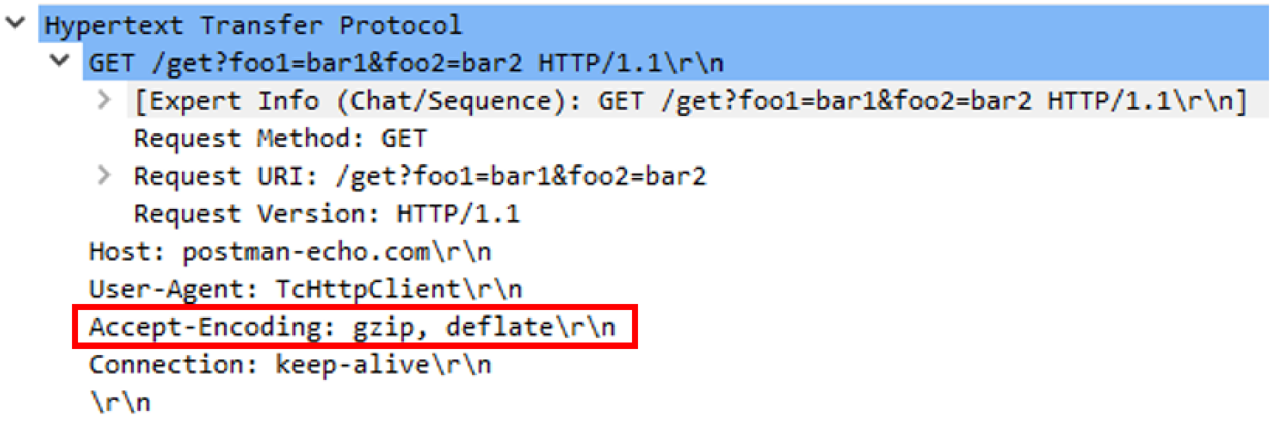
4.4 Compression

In general, the term data compression refers to the ability to reduce the number of bits needed to represent data. One way of dealing with this is to provide recurring strings with a reference to the first of these strings through a compression algorithm. Appropriate compression must occur without loss of information.

Two compression types that are very common in HTTP communication are gzip and deflate. Both have been supported in transmit and receive direction in the HTTP driver and the Tc3_IotBase library since TwinCAT version 3.1.4024.11. The methods belong to the group of lossless compression types.

In TwinCAT the compression can be set individually for a FB_IotHttpRequest [▶ 25] function block and is passed to it as an input parameter. By default, no compression is used in transmit direction. In receive direction the driver can receive and correctly represent both types of compression without separate settings.

The following diagram shows an HTTP request to the Postman Echo test REST API (see PostmanEcho [▶ 126]). The TwinCAT HTTP client notifies the REST API via a header field that both deflate and gzip are supported in receive direction.



```
Hypertext Transfer Protocol
  GET /get?foo1=bar1&foo2=bar2 HTTP/1.1\r\n
    [Expert Info (Chat/Sequence): GET /get?foo1=bar1&foo2=bar2 HTTP/1.1\r\n]
    Request Method: GET
    Request URI: /get?foo1=bar1&foo2=bar2
    Request Version: HTTP/1.1
  Host: postman-echo.com\r\n
  User-Agent: TcHttpClient\r\n
  Accept-Encoding: gzip, deflate\r\n
  Connection: keep-alive\r\n
  \r\n
```

4.5 Security

When considering protection of data communication, a distinction can be made between two levels: protection of the transport channel (Transport layer [▸ 15]) and protection at Application level [▸ 19]. Both are described below.

4.5.1 Transport layer

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

Contents of this chapter

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

Cipher suite definition

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

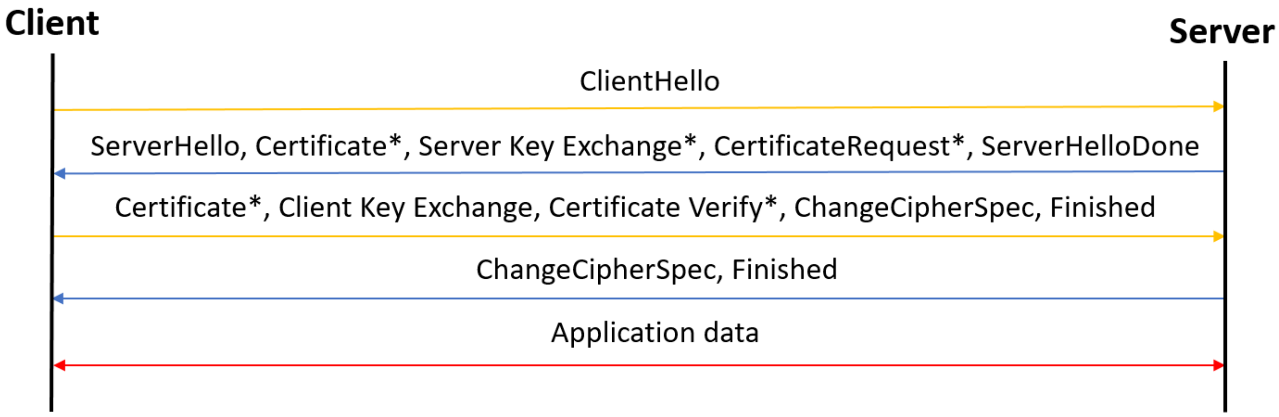
TLS communication flow

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

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

- The server identifies itself through a certificate (see Server certificate [▸ 16])
- Client and server identify themselves through a certificate (see Client/Server certificate [▸ 17])
- Pre-shared keys (see Pre-shared keys [▸ 18])

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


Client
Server
ClientHello
ServerHello, Certificate*, Server Key Exchange*, CertificateRequest*, ServerHelloDone
Certificate*, Client Key Exchange, Certificate Verify*, ChangeCipherSpec, Finished
ChangeCipherSpec, Finished
Application data

Exemplary explanation based on RSA

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

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

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

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

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

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

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

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

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

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

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

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

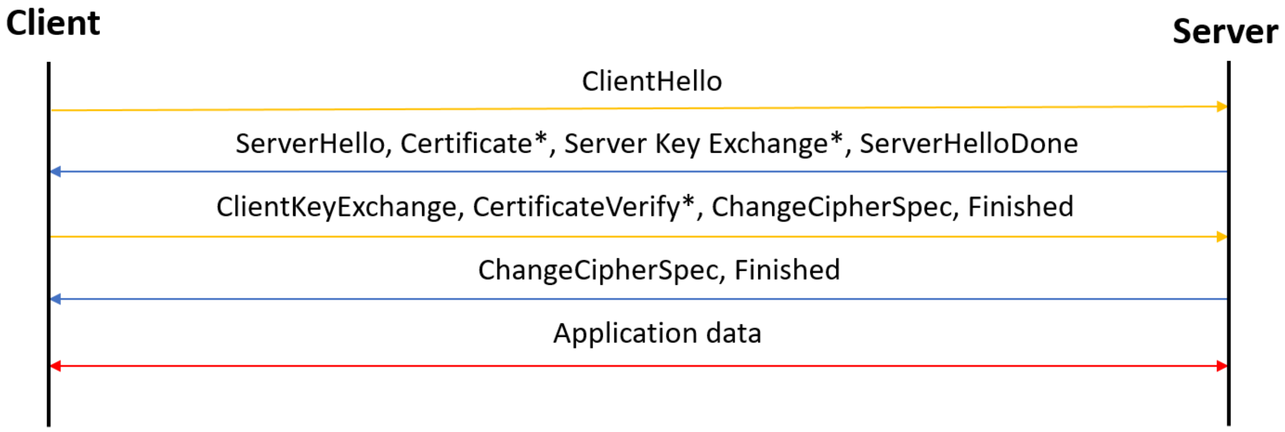
Change Cipher Spec: The server decrypts the pre-master secret that the client encrypted with its public key. Since only the server has its private key, only the server can decrypt this pre-master secret. This ensures that the symmetric key is only known to the client and the server. The server then calculates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

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

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

4.5.1.1 Server certificate

This section covers a situation where the client wants to verify the server certificate but the server does not want to verify the client certificate. In this case the communication flow described in chapter Transport layer [▶ 15] is shortened as follows.


Client
Server
ClientHello
ServerHello, Certificate*, Server Key Exchange*, ServerHelloDone
ClientKeyExchange, CertificateVerify*, ChangeCipherSpec, Finished
ChangeCipherSpec, Finished
Application data

Verification of the server certificate

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

Application in TwinCAT

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

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

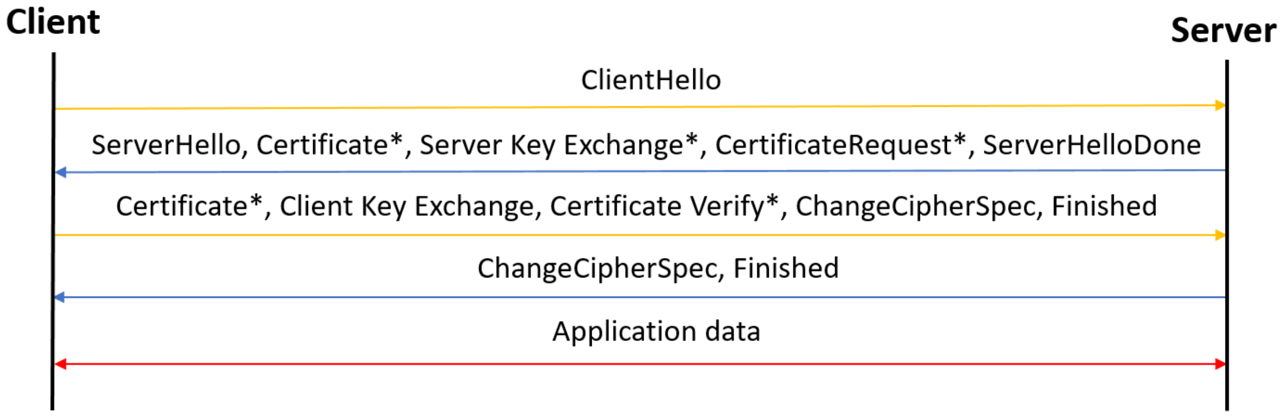
```
fbClient.stTLS.sCA:= 'C:\TwinCAT\3.1\Config\Certificates\someCA.pem';
```

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

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

4.5.1.2 Client/Server certificate

This section considers the case where both the client certificate and the server certificate are verified. The slightly modified communication flow (compared to the Server certificate [▶ 16] chapter) is visualized in the following diagram. The individual steps of the TLS connection establishment are described in chapter Transport layer [▶ 15].


Client
Server
ClientHello
ServerHello, Certificate*, Server Key Exchange*, CertificateRequest*, ServerHelloDone
Certificate*, Client Key Exchange, Certificate Verify*, ChangeCipherSpec, Finished
ChangeCipherSpec, Finished
Application data

Application in TwinCAT

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

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

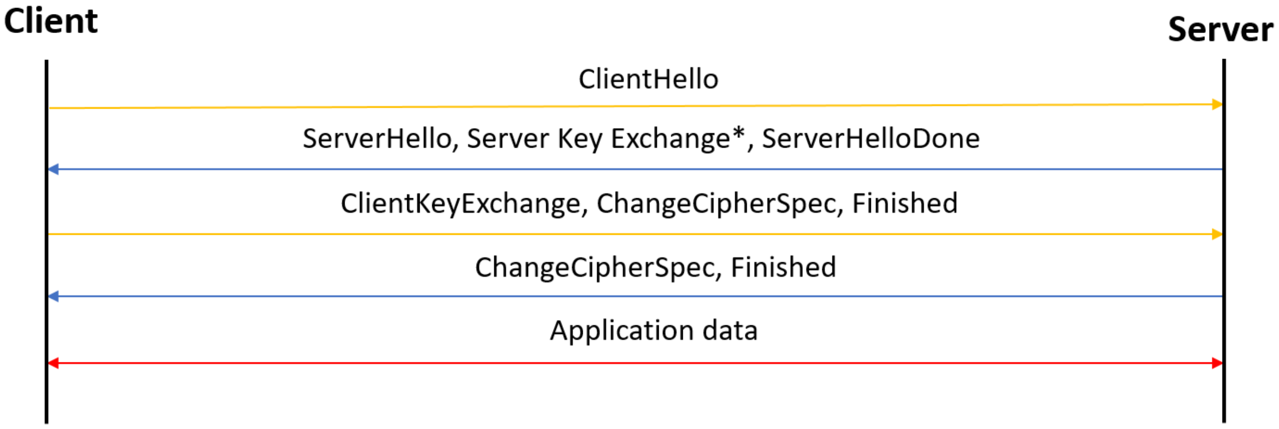
```
fbClient.stTLS.sCA:= 'C:\TwinCAT\3.1\Config\Certificates\someCA.pem';
fbClient.stTLS.sCert:= 'C:\TwinCAT\3.1\Config\Certificates\someCRT.pem';
fbClient.stTLS.sKeyFile:= 'C:\TwinCAT\3.1\Config\Certificates\someprivatekey.pem.key';
fbClient.stTLS.sKeyPwd:= 'yourkeyfilepasswordhere';
```

In principle, the user can also disable the validation of the server certificate in the case described. However, this is associated with weaker security (see Server certificate [▶ 16])

4.5.1.3 Pre-shared keys

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

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


Client
Server
ClientHello
ServerHello, Server Key Exchange*, ServerHelloDone
ClientKeyExchange, ChangeCipherSpec, Finished
ChangeCipherSpec, Finished
Application data

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

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

Application in TwinCAT

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

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

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

4.5.2 Application level

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

4.5.2.1 JSON Web Token (JWT)

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

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

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

4.5.2.2 AWS Signature Version 4

AWS Signature V4 adds authentication information to requests to AWS services (for example instantiating a virtual machine via the EC2 service) sent via the HTTP protocol. For security reasons, most requests to AWS services must be signed with an access key. This access key, consisting of an access key ID and a secret access key, can be set up and managed in a corresponding AWS account. For further information please refer to the AWS documentation.

4.6 Re-parameterization

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

Example use cases for a new parameterization: replacement of a token that is about to expire, certificates that need to be renewed or a changed IP address of the HTTP server. The following section describes the procedure for re-parameterization of an HTTP client that has already been parameterized.

For cyclic background communication of an HTTP client function block with the TwinCAT driver, the Execute method must be called cyclically in the background (see FB_IotHttpClient [▶ 22]). When a program is started, calling this method first sets the parameters of the HTTP client instance, after which it is possible to send HTTP requests. The variable bConfigured is able to detect completion of this parameterization. The HTTP client is fully configured once the variable has the value TRUE.

To carry out a new parameterization, the call of the Execute method must be suspended and the Disconnect method must be called. It is important to ensure that no more HTTP requests are sent at the time of the call. By calling the Disconnect method bConfigured is set to FALSE. The parameters are then reset and the Execute method is called again as usual. The following code snippet shows a simple re-parameterization using a trigger.

```
IF bReset THEN
    fbHttpClient.Disconnect();
    IF fbHttpClient.bConfigured=FALSE THEN
        fbHttpClient.sHostName:='postman-echo.com';
        fbHttpClient.stTLS.sCA:='C:\TwinCAT\3.1\Config\Certificates\caCERT.cer';
        bReset:=FALSE;
    END_IF
ELSE
    fbHttpClient.Execute();
END_IF
```

The trigger variable bReset suspends the call of the Execute method and calls the Disconnect method. Once bConfigured is set to FALSE on the HTTP client function block, the host name and CA certificate parameters are reset. bReset is then reset and the Excute method is called again.

4.7 URL redirects

With a server-side URL redirect a server indicates to a client that a requested server resource has been moved permanently or temporarily. Such redirects are not automatically evaluated in the IoT driver. It is therefore left to the application logic to evaluate a URL redirect.

Example: HTTP status code 301 ("Moved permanently"). This is where a server notifies a client that a server resource has been moved permanently. A header field with the name "Location" is added to the HTTP response, so that the client can be notified of the new address of the resource. This header field then contains the new URL where the resource is located.

In a TwinCAT application, after receiving an HTTP response via the status code query, an application logic could be implemented that extracts the new URL from a response with status code 301 and sends a new request with the updated URL to the server. To do this, the "Location" header field of the response must be queried and used as the target URL for a new request.

4.8 Cookies

Cookies are used in the HTTP environment for three use cases. The use cases include session management, which is used for logins or generally for everything that the server has to "remember". Cookies are also used for personalization and tracking.

From a technical point of view, when an HTTP server responds to the client it sends back one or more header fields named "Set-Cookie" in the response. These cookies will then be included in future requests to the HTTP server, although in this case separated by a semicolon in a single header field.

In TwinCAT the header fields described above are automatically combined with the cookies into one header field and separated by a line feed. This does not only apply to header fields of the type "Set-Cookie", but in all cases where an HTTP response of the server contains several header fields of the same name. The following sample illustrates this.

```
HTTP/1.1 200 OK
Content-Type: text/html
Set-Cookie: token1=Beckhoff
Set-Cookie: token2=IoT
```

If the response is received by the TwinCAT IoT driver, the two header fields with the name "Set-Cookie" are automatically merged into one. The following string is then output after querying the header field in the program code:

```
'token1=Beckhoff$Ntoken2=IoT';
```

5 SPS API

5.1 Tc3_IotBase

5.1.1 FB_IotHttpAwsSigV4HeaderFieldMap

This function block enables signing of a header with AWS Signature Version 4 (see AWS Signature Version 4 [▶ 19]).

Syntax

Definition:

```
FUNCTION BLOCK FB_IotHttpAwsSig4HeaderFieldMap EXTENDS FB_IotHttpHeaderFieldMap
VAR_OUTPUT
    bError      : BOOL;
    hrErrorCode : HRESULT;
END_VAR
```

Outputs

Name	Type	Description
bError	BOOL	Becomes TRUE when an error situation occurs
hrErrorCode	HRESULT	Returns an ADS return code. An explanation of the possible ADS return codes can be found in the Appendix.

Methods

Name	Description
AddField [▶ 24]	Adds a header field to an instance of this function block.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4024.12 or higher	IPC or CX (x86, x64, ARM)	Tc3_IotBase

5.1.1.1 SetParameter

This method sets the required parameters on a header intended for requests to AWS services and signs it. The individual parameters must be sorted alphabetically for signing. This is done internally by the driver, so the user does not have to pay attention to the order.

Syntax

```
METHOD SetParameter : BOOL
VAR_IN_OUT CONSTANT
    service : STRING;
    region : STRING;
    accessKey : STRING;
    secretKey : STRING;
    signedHeaders : STRING;
END_VAR
```

Inputs

Name	Type	Description
service	STRING	AWS service code
region	STRING	AWS data center code.
accessKey	STRING	The AccessKey of the two-part access key of an AWS IAM user. Required for authentication of requests.
secretKey	STRING	The SecretKey of the two-part access key of an AWS IAM user. Required for authentication of requests.
signedHeaders	STRING	Describes a semicolon-separated list of HTTPS headers to include in the signature. For further information please refer to the AWS documentation.

Return value

Name	Type	Description
SetParameter	BOOL	Returns TRUE if the method call was successful.

5.1.2 FB_IotHttpClient

This function block enables communication with a HTTP server.

A client function block is responsible for the connection to exactly one server. The Execute() method of the function block must be called cyclically in order to enable the functionality of the HTTP client. The exact resource on the server is specified with the sUri property of the FB_IotHttpRequest function block.

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

Syntax

Definition:

```
FUNCTION BLOCK FB_IotHttpClient
VAR_INPUT
    //default is local host
    sHostName          : STRING((ParameterList.cSizeOfHttpClientHostName-1)) :=’127.0.0.1’;
    //default is 80
    nHostPort          : UINT :=80;
    bKeepAlive         : BOOL :=TRUE;
    tConnectionTimeout : TIME :=T#10s;
    //optional parameter
    stTLS              : ST_IotSocketTls;
END_VAR
VAR_OUTPUT
    bError             : BOOL;
    hrErrorCode        : HRESULT;
    bConfigured        : BOOL;
    //TRUE if connection to server is established
    bConnected         : BOOL;
END_VAR
```

Inputs

Name	Type	Description
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 value is 80.
bKeepAlive	BOOL	Specifies if the connection to the server should be kept until either server or client terminate it. Please keep in mind, that not every server supports this feature.
tConnectionTimeout	TIME	Specifies the connection timeout.
stTLS	ST_IotSocketTls [▶ 30]	If the server offers a TLS-secured connection, the required configuration can be implemented here.

Outputs

Name	Type	Description
bError	BOOL	Becomes TRUE as soon as an error situation occurs.
hrErrorCode	HRESULT	Returns an ADS return code. An explanation of the possible ADS return codes can be found in the Appendix.
bConfigured	BOOL	Becomes TRUE when the initial configuration is finished.
bConnected	BOOL	Becomes TRUE if a connection to the host is established. Please keep in mind, that the connection is usually closed after every request, depending on the input parameter bKeepAlive.

Methods

Name	Description
Disconnect	Disconnects the client from the server, when there is an active connection.
Execute	Method for background communication with the TwinCAT driver. The method must be called cyclically for every function block instance.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4024.7 or higher	IPC or CX (x86, x64, ARM)	Tc3_IotBase

5.1.3 FB_IotHttpHeaderFieldMap

This function block gives the possibility to add additional header fields to a HTTP request. For that the method AddField has to be called on this function block. Afterwards the function block has to be passed to the HTTP command that is send to the server.

Syntax

Definition:

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

Outputs

Name	Type	Description
bError	BOOL	Becomes TRUE as soon as an error situation occurs.
hrErrorCode	HRESULT	Returns an ADS return code. An explanation of the possible ADS return codes can be found in the Appendix.

Methods

Name	Description
AddField [▶ 24]	Adds a header field to an instance of this function block.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4024.7 or higher	IPC or CX (x86, x64, ARM)	Tc3_IotBase

5.1.3.1 AddField

This method adds an additional header field to an instance of the FB_IotHttpHeaderFieldMap function block.

Syntax

```
METHOD AddField : BOOL
VAR_IN_OUT CONSTANT
    sField       : STRING;
    sValue       : STRING;
END_VAR
VAR_INPUT
    bAppendValue : BOOL;
END_VAR
```

Inputs

Name	Type	Description
sField	STRING	Name of the field that is added to the header of the HTTP request.
sValue	STRING	Value of the added header field.
bAppendValue	BOOL	Replaces existing header field value (FALSE) or appends the value to the existing one (TRUE).

Return value

Name	Type	Description
AddField	BOOL	Returns TRUE when the method call was successful.

5.1.3.2 FindField

This method searches for a specific header field in an instance of the function block FB_IotHttpHeaderFieldMap.

Syntax

```
METHOD FindField : BOOL
VAR_IN_OUT CONSTANT
    sField       : STRING;
END_VAR
VAR_INPUT
    pValue       : PVOID;
    nValueSize   : UDINT;
END_VAR
```

Inputs

Name	Type	Description
sField	STRING	Name of the field to be found in the header.
pValue	PVOID	Pointer to the data memory where the found header field is to be stored.
nValueSize	UDINT	The size of the memory variables.

Return value

Name	Type	Description
FindField	BOOL	Returns TRUE if the method call was successful.

5.1.4 FB_IotHttpRequest

This function block enables the PLC to send HTTP commands to the server that has been configured with the function block FB_IotHttpClient. To be able to process an HTTP response, the output bBusy of this function block must first return FALSE. Then the content payload, specific header fields or a JSON string can be extracted from the response.

Syntax

Definition:

```
FUNCTION_BLOCK FB_IotHttpRequest IMPLEMENTS ITcIotHttpResponse
VAR_INPUT
    sContentType     : STRING((ParameterList.cSizeOfHttpContentType-1)) :='text/html;charset=UTF-8';
    eCompressionMode : E_IotHttpCompressionMode                         := 'E_IotHttpCompressionMode
.NoCompression';
END_VAR
VAR_OUTPUT
    bBusy            : BOOL;
    bError           : BOOL;
    eErrorId         : ETcIotHttpRequestError;
    nStatusCode      : UINT;
    nContentSize     : UDINT;
END_VAR
```

Inputs

Name	Type	Description
sContentType	STRING	Content type of the request, e.g. the initial value "text/html; charset=UTF-8".
eCompressionMode	E_IotHttpCompressionMode	Enumeration of the different compression modes. Further information is provided in the chapter Compression [▸ 14]. The list can be found in the Appendix.

Outputs

Name	Type	Description
bBusy	BOOL	This output remains TRUE until the function block has executed the command.
bError	BOOL	Becomes TRUE as soon as an error situation occurs.
eErrorId	ETcIotHttpRequestError	Enumeration of status code on transport level e.g. if a certificate validation has failed. The enumeration can be found in the Appendix.
nStatusCode	UINT	HTTP status code send from the server within the response.
nContentSize	UDINT	Size of the content payload.

Methods

Name	Description
GetContent [▸ 26]	Reads the response from the HTTP server.
GetHeaderField [▸ 27]	Reads the value from a specified header field.
GetJsonDomContent [▸ 28]	Parses the server response as JSON object.
SendJsonDomRequest [▸ 28]	Sends a request to the server as JSON object.
SendRequest [▸ 29]	Sends a HTTP command to the server.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4024.7 or higher	IPC or CX (x86, x64, ARM)	Tc3_IotBase

5.1.4.1 GetContent

This method reads the HTTP response and stores it into a variable. To be able to get the content out of a HTTP response, the bBusy output of the FB_IotHttpRequest function block has to return FALSE.

Syntax

```
METHOD GetContent : BOOL
VAR_INPUT
    pContent             : PVOID;
    nContentSize         : UDINT;
    bSetNullTermination : BOOL;
END_VAR
```

Inputs

Name	Type	Description
pContent	PVOID	Pointer to the data memory, where the read data should be stored.
nContentSize	UDINT	Size of content payload.
bSetNullTermination	BOOL	Specifies if the payload should be read with null termination or without null termination.

Return value

Name	Type	Description
GetContent	BOOL	Returns TRUE if the method call was successful.

5.1.4.2 GetHeaderField

This method is able to read the value of a specified header field from a HTTP response. To be able to get a header field out of a HTTP response, the bBusy output of the FB_IotHttpRequest function block has to return FALSE.

Syntax

```
METHOD GetHeaderField : BOOL
VAR_IN_OUT CONSTANT
    sField       : STRING;
END_VAR
VAR_INPUT
    pValue       : PVOID;
    nValueSize   : UDINT;
END_VAR
```

Inputs

Name	Type	Description
sField	STRING	Name of the specified header field.
pValue	PVOID	Pointer to the data memory, where the read data should be stored.
nValueSize	UDINT	The maximum size that the buffer of this method can have.

Return value

Name	Type	Description
GetHeaderField	BOOL	Returns TRUE if the method call was successful.

5.1.4.3 GetJsonDomContent

This method parses the response into a JSON object. When the server always responds with a JSON object, the step of converting a string to a JSON object is skipped. To be able to get the content of a HTTP response, the bBusy output of the FB_IotHttpRequest function block has to return FALSE.

Syntax

```
METHOD GetJsonDomContent : SJsonValue
VAR_INPUT
    fbJson : REFERENCE TO FB_JsonDomParser;
END_VAR
```

Inputs

Name	Type	Description
fbJson	REFERENCE TO FB_JsonDomParser	The instance of the JsonDomParser which should be used for parsing the JSON object.

Return value

Name	Type	Description
GetJsonDomContent	SJsonValue	JSON root node to start parsing.

5.1.4.4 SendJsonDomRequest

This method sends a request with a JSON object payload to a HTTP server. This method can e.g. be used when the server only accepts JSON object payloads.

Syntax

```
METHOD SendJsonDomRequest : BOOL
VAR_IN_OUT CONSTANT
    sUri         : STRING;
END_VAR
VAR_INPUT
    fbClient     : REFERENCE TO FB_IotHttpClient;
    eRequestType : ETcIotHttpRequestType;
    fbJson       : REFERENCE TO FB_JsonDomParser;
    fbHeader     : REFERENCE TO FB_IotHttpHeaderFieldMap;
END_VAR
```

Inputs

Name	Type	Description
sUri	STRING	The identifier of the resource on the host. Please note, that the sUri is appended to the host name of the IotHttpClient.
fbClient	REFERENCE TO FB_IotHttpClient	The HTTP client function block instance from which the request should be send.
eRequestType	ETcIotHttpRequestType	The type of HTTP command the request should have. The enumeration can be found in the Appendix.
fbJson	REFERENCE TO FB_JsonDomParser	The instance of JsonDomParser for parsing a string to a JSON object.
fbHeader	REFERENCE TO FB_IotHttpHeaderFieldMap	The instance of IotHttpHeaderFieldMap to edit the header of the HTTP request.

Return value

Name	Type	Description
SendJsonDomRequest	BOOL	Returns TRUE when the method call was successful.

5.1.4.5 SendRequest

This method sends a request to a HTTP server.

Syntax

```
METHOD SendRequest : BOOL
VAR_IN_OUT CONSTANT
    sUri         : STRING;
END_VAR
VAR_INPUT
    fbClient     : REFERENCE TO FB_IotHttpClient;
    eRequestType : ETcIotHttpRequestType;
    pContent     : PVOID;
    nContentSize : UDINT;
    fbHeader     : REFERENCE TO FB_IotHttpHeaderFieldMap;
END_VAR
```

Inputs

Name	Type	Description
sUri	STRING	The identifier of the resource on the host. Please note, that the sUri is appended to the host name of the FB_IotHttpClient.
fbClient	REFERENCE TO FB_IotHttpClient	The HTTP client function block instance from which the request should be sent.
eRequestType	ETcIotHttpRequestType	The type of HTTP command the request should have. The enumeration can be found in the Appendix.
pContent	PVOID	Pointer to the data memory, from where the request payload should be read.
nContentSize	UDINT	Size of the request payload.
fbHeader	REFERENCE TO FB_IotHttpHeaderFieldMap	The instance of the FB_IotHttpHeaderFieldMap to edit the header of the HTTP request.

Return value

Name	Type	Description
SendRequest	BOOL	Returns TRUE if the method call was successful.

5.1.5 ST_IotSocketTls

The following type contains the TLS security settings for the HTTP client.
Either CA (certificate authority) or PSK (PreSharedKey) can be used.

Syntax

Definition:

```
TYPE ST_IotSocketTls :
STRUCT
    sCA                : STRING(255*);
    sCert              : STRING(255*);
    sKeyFile           : STRING(255*);
    sKeyPwd            : STRING(255*);
    sCrl               : STRING(255*);
    sCiphers           : STRING(255*);
    sVersion           : STRING(80) := 'tlsv1.2';
    bNoServerCertCheck : BOOL := FALSE;

    sPskIdentity       : STRING(255*);
    aPskKey            : ARRAY[1..64*] OF BYTE;
    nPskKeyLen         : USINT;
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 authentification at the server
sKeyFile	STRING(255)	Private key of the client
sKeyPwd	STRING(255)	Password of the private key file, if applicable
sCrl	STRING(255)	Path to the certificate revocation list, which may be present in PEM or DER format
sCiphers	STRING(255)	Cipher suites to be used, specified in OpenSSL string format
sVersion	STRING(80)	TLS version to be used
bNoServerCertCheck	BOOL	Disables verification of the server certificate validity
sPskIdentity	STRING(255)	PreSharedKey identity for TLS PSK connection
aPskKey	ARRAY[1..64] OF BYTE	PreSharedKey for TLS PSK connection
nPskKeyLen	USINT	Lenght of the PreSharedKey in bytes

All strings and arrays that have been tagged with a *, are initialized with the value in the brackets. These values can be accessed and changed trough the parameter list. This is not possible during the runtime, only before compiling the code.

5.1.6 ParameterList

The parameter list of the Tc3_IotBase library can be used to modify the size of different parameters. This modification is not possible during runtime and can only be done before compiling the code. For a better overview, only the parameter that are important for HTTP are listed in this documentation.

Syntax

Definition:

```
VAR_GLOBAL CONSTANT Parameter_List
    cSizeOfHttpContentType      : UDINT(32..10000);
    cSizeOfHttpClientHostName   : UDINT(32..10000);
    cSizeOfHttpTlsFilePaths     : UDINT(32..10000);
    cSizeOfHttpTlsKeyFile       : UDINT(32..10000);
    cSizeOfHttpTlsKeyPwd        : UDINT(32..10000);
    cSizeOfHttpTlsCiphers       : UDINT(32..10000);
    cSizeOfHttpTlsPskIdentity   : UDINT(32..10000);
    cSizeOfHttpTlsPskKey        : UDINT(32..10000);
END_VAR
```

Parameter

Name	Type	Description
cSizeOfHttpContentType	UDINT(32..10000)	String size definition of HTTP content type (in bytes).
cSizeOfHttpClientHostName	UDINT(32..10000)	String size definition of HTTP client hostname (in bytes).
cSizeOfHttpTlsFilePaths	UDINT(32..10000)	String size definition of HTTP TLS file path (in bytes).
cSizeOfHttpTlsKeyFile	UDINT(32..10000)	String size definition of HTTP TLS key file (in bytes).
cSizeOfHttpTlsKeyPwd	UDINT(32..10000)	String size definition of HTTP TLS private key password (in bytes).
cSizeOfHttpTlsCiphers	UDINT(32..10000)	String size definition of HTTP TLS cipher suites (in bytes).
cSizeOfHttpTlsPskIdentity	UDINT(32..10000)	String size definition of HTTP TLS PSK identity (in bytes).
cSizeOfHttpTlsPskKey	UDINT(32..10000)	String size definition of HTTP TLS PSK (in bytes).

5.2 Tc3_JsonXml

5.2.1 Function blocks

5.2.1.1 FB_JsonDomParser

This function block is derived from the same internal function block as FB_JsonDynDomParser [▶ 62] and thus offers the same interface.

The two derived function blocks differ only in their internal memory management. FB_JsonDomParser is optimized for the fast and efficient parsing and creation of JSON documents that are only changed a little. The function block FB_JsonDynDomParser [▶ 62] is recommended for JSON documents to which many changes are made.

⚠ WARNING

Use of router memory

The function block occupies new memory with each change, e.g. with the methods SetObject() or SetJson(). As a result, the amount of router memory used can grow enormously after repeated actions. This allocated memory is only released again by calling the method NewDocument [▶ 51]().

Strings in UTF-8 format

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

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

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

Further information on the UTF-8 STRING format and available display and conversion options can be found in the documentation for the Tc2_Utilities PLC library.

Requirements

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

Also see about this

- ParseDocument [▸ 51]

5.2.1.1.1 AddArrayMember

This method adds an array member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.2 AddBase64Member

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

Syntax

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

Sample call:

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

5.2.1.1.3 AddBoolMember

This method adds a Bool member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.4 AddDateTimeMember

This method adds a DateTime member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.5 AddDcTimeMember

This method adds a DcTime member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.6 AddDoubleMember

This method adds a Double member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.7 AddFileTimeMember

This method adds a FileTime member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.8 AddHexBinaryMember

This method adds a HexBinary member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.9 AddInt64Member

This method adds an Int64 member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.10 AddIntMember

This method adds an Int member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.11 AddJsonMember

This method adds a JSON member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.12 AddNullMember

This method adds a NULL member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.13 AddObjectMember

This method adds an Object member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.14 AddStringMember

This method adds a String member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.15 AddUint64Member

This method adds an UInt64 member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.16 AddUintMember

This method adds an UInt member to a JSON object.

Syntax

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

Sample call:

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

5.2.1.1.17 ArrayBegin

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

Syntax

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

Sample call:

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

5.2.1.1.18 ArrayEnd

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

Syntax

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

Sample call:

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

5.2.1.1.19 ClearArray

This method deletes the content of an array.

Syntax

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

Sample call:

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

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

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

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

5.2.1.1.20 CopyDocument

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

Syntax

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

Sample call:

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

5.2.1.1.21 CopyJson

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

Syntax

```
METHOD CopyJson : UDINT
VAR_INPUT
  v : 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","meta":{"batteryVoltage":"1547mV","clickType":"SINGLE"}}';
```

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

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

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

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

5.2.1.1.22 CopyString

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

Syntax

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

Sample call:

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

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

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

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

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

```
SINGLE
```

5.2.1.1.23 FindMember

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

Syntax

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

Sample call:

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

5.2.1.1.24 FindMemberPath

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

Syntax

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

Sample call:

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

5.2.1.1.25 GetArraySize

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

Syntax

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

Sample call:

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

5.2.1.1.26 GetArrayValue

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

Syntax

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

Sample call:

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

5.2.1.1.27 GetArrayValueByIdx

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

Syntax

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

Sample call:

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

5.2.1.1.28 GetBase64

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

Syntax

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

Sample call:

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

5.2.1.1.29 GetBool

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

Syntax

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

5.2.1.1.30 GetDateTime

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

Syntax

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

5.2.1.1.31 GetDcTime

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

Syntax

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

Sample call:

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

5.2.1.1.32 GetDocument

This method returns the content of the DOM memory as the data type STRING(255). With longer strings, the method will return a NULL string. In this case the method CopyDocument [▶ 38]() must be used.

Syntax

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

Sample call:

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

5.2.1.1.33 GetDocumentLength

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

Syntax

```
METHOD GetDocumentLength: UDINT
```

Sample call:

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

5.2.1.1.34 GetDocumentRoot

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

Syntax

```
METHOD GetDocumentRoot : SJsonValue
```

Sample call:

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

5.2.1.1.35 GetDouble

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

Syntax

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

5.2.1.1.36 GetFileTime

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

Syntax

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

Sample call:

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

5.2.1.1.37 GetHexBinary

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

Syntax

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

Sample call:

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

5.2.1.1.38 GetInt

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

Syntax

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

5.2.1.1.39 GetInt64

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

Syntax

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

5.2.1.1.40 GetJson

This method returns the value of a property as data type STRING(255), if this is a JSON document itself. With longer strings, the method will return a NULL string. In this case the method CopyJson [▶ 39]() must be used.

Syntax

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

Sample call:

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

5.2.1.1.41 GetJsonLength

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

Syntax

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

Sample call:

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

5.2.1.1.42 GetMaxDecimalPlaces

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

Syntax

```
METHOD GetMaxDecimalPlaces : DINT
```

Sample call:

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

5.2.1.1.43 GetMemberName

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

Syntax

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

Sample call:

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

5.2.1.1.44 GetMemberValue

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

Syntax

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

Sample call:

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

5.2.1.1.45 GetString

This method returns the value of a property of the data type STRING(255). With longer strings, the method will return a NULL string. In this case the method CopyString [▶ 39]() must be used.

Syntax

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

5.2.1.1.46 GetStringLength

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

Syntax

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

Sample call:

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

5.2.1.1.47 GetType

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

Syntax

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

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

Sample call:

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

5.2.1.1.48 GetUint

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

Syntax

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

5.2.1.1.49 GetUint64

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

Syntax

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

5.2.1.1.50 HasMember

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

Syntax

```
METHOD HasMember : BOOL
VAR_INPUT
  v : SJsonValue;
```

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

Sample call:

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

5.2.1.1.51 IsArray

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

Syntax

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

5.2.1.1.52 IsBase64

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

Syntax

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

5.2.1.1.53 IsBool

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

Syntax

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

5.2.1.1.54 IsDouble

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

Syntax

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

5.2.1.1.55 IsFalse

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

Syntax

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

5.2.1.1.56 IsHexBinary

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

Syntax

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

Sample call:

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

5.2.1.1.57 IsInt

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

Syntax

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

5.2.1.1.58 IsInt64

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

Syntax

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

5.2.1.1.59 IsISO8601TimeFormat

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

Syntax

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

5.2.1.1.60 IsNull

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

Syntax

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

5.2.1.1.61 IsNumber

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

Syntax

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

5.2.1.1.62 IsObject

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

Syntax

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

5.2.1.1.63 IsString

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

Syntax

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

5.2.1.1.64 IsTrue

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

Syntax

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

5.2.1.1.65 IsUint

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

Syntax

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

5.2.1.1.66 IsUint64

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

Syntax

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

5.2.1.1.67 LoadDocumentFromFile

This method loads a JSON document from a file.

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

Syntax

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

Sample call:

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

5.2.1.1.68 MemberBegin

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

Syntax

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

Sample call:

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

5.2.1.1.69 MemberEnd

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

Syntax

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

Sample call:

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

5.2.1.1.70 NewDocument

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

Syntax

```
METHOD NewDocument : SJsonValue
```

Sample call:

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

5.2.1.1.71 NextArray

5.2.1.1.72 ParseDocument

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

Syntax

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

Sample call:

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

5.2.1.1.73 PushbackBase64Value

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

Syntax

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

Sample call:

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

5.2.1.1.74 PushbackBoolValue

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

Syntax

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

Sample call:

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

5.2.1.1.75 PushbackDateTimeValue

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

Syntax

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

Sample call:

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

5.2.1.1.76 PushbackDcTimeValue

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

Syntax

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

Sample call:

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

5.2.1.1.77 PushbackDoubleValue

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

Syntax

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

Sample call:

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

5.2.1.1.78 PushbackFileTimeValue

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

Syntax

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

Sample call:

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

5.2.1.1.79 PushbackHexBinaryValue

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

Syntax

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

Sample call:

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

5.2.1.1.80 PushbackInt64Value

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

Syntax

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

Sample call:

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

5.2.1.1.81 PushbackIntValue

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

Syntax

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

Sample call:

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

5.2.1.1.82 PushbackJsonValue

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

Syntax

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

Sample call:

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

5.2.1.1.83 PushbackNullValue

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

Syntax

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

Sample call:

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

5.2.1.1.84 PushbackStringValue

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

Syntax

```
METHOD PushbackStringValue : SJsonValue
VAR_INPUT
  v : SJsonValue;
```

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

Sample call:

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

5.2.1.1.85 PushbackUint64Value

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

Syntax

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

Sample call:

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

5.2.1.1.86 PushbackUintValue

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

Syntax

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

Sample call:

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

5.2.1.1.87 RemoveAllMembers

This method removes all child elements from a given property.

Syntax

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

Sample call:

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

5.2.1.1.88 RemoveArray

This method deletes the value of the current array iterator.

Syntax

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

Sample call:

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

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

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

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

5.2.1.1.89 RemoveMember

This method deletes the property at the current iterator.

Syntax

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

Sample call:

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

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

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

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

5.2.1.1.90 RemoveMemberByName

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

Syntax

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

Sample call:

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

5.2.1.1.91 SaveDocumentToFile

This method saves a JSON document in a file.

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

Syntax

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

Sample call:

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

5.2.1.1.92 SetArray

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

Syntax

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

Sample call:

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

5.2.1.1.93 SetBase64

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

Syntax

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

Sample call:

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

5.2.1.1.94 SetBool

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

Syntax

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

Sample call:

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

5.2.1.1.95 SetDateTime

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

Syntax

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

Sample call:

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

5.2.1.1.96 SetDcTime

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

Syntax

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

Sample call:

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

5.2.1.1.97 SetDouble

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

Syntax

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

Sample call:

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

5.2.1.1.98 SetFileTime

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

Syntax

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

Sample call:

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

5.2.1.1.99 SetHexBinary

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

Syntax

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

Sample call:

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

5.2.1.1.100 SetInt

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

Syntax

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

Sample call:

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

5.2.1.1.101 SetInt64

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

Syntax

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

Sample call:

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

5.2.1.1.102 SetJson

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

Syntax

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

Sample call:

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

5.2.1.1.103 SetMaxDecimalPlaces

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

Syntax

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

Sample call:

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

5.2.1.1.104 SetNull

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

Syntax

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

Sample call:

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

5.2.1.1.105 SetObject

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

Syntax

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

Sample call:

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

5.2.1.1.106 SetString

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

Syntax

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

Sample call:

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

5.2.1.1.107 SetUint

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

Syntax

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

Sample call:

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

5.2.1.1.108 SetUint64

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

Syntax

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

Sample call:

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

5.2.1.2 FB_JsonDynDomParser

This function block is derived from the same internal function block as FB_JsonDomParser [▶ 32] and thus offers the same interface.

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

Strings in UTF-8 format

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

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

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

Further information on the UTF-8 STRING format and available display and conversion options can be found in the documentation for the Tc2_Utilities PLC library.

Requirements

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

5.2.1.3 FB_JsonSaxReader

Requirements

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

5.2.1.3.1 DecodeBase64

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

Syntax

```
METHOD DecodeBase64 : BOOL
VAR_INPUT
  sBase64 : STRING;
  pBytes  : POINTER TO BYTE;
  nBytes  : REFERENCE TO DINT;
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

Sample call:

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

5.2.1.3.2 DecodeDateTime

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

Syntax

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

Sample call:

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

5.2.1.3.3 DecodeDcTime

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

Syntax

```
METHOD DecodeDcTime : BOOL
VAR_IN_OUT CONSTANT
  sDC : STRING;
END_VAR
VAR_OUTPUT
  nDC : DCTIME;
  hrErrorCode : HRESULT;
END_VAR
```

Sample call:

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

5.2.1.3.4 DecodeFileTime

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

Syntax

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

Sample call:

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

5.2.1.3.5 DecodeHexBinary

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

Syntax

```
METHOD DecodeHexBinary : BOOL
VAR_IN_OUT CONSTANT
  sHex : STRING;
END_VAR
VAR_INPUT
  pBytes : POINTER TO BYTE;
  nBytes : REFERENCE TO DINT;
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

Sample call:

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

5.2.1.3.6 IsBase64

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

Syntax

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

Sample call:

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

5.2.1.3.7 IsHexBinary

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

Syntax

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

Sample call:

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

5.2.1.3.8 IsISO8601TimeFormat

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

Syntax

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

Sample call:

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

5.2.1.3.9 Parse

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

Syntax

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

5.2.1.3.10 ParseValues

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

Syntax

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

5.2.1.4 FB_JsonSaxWriter

Strings in UTF-8 format

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

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

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

Further information on the UTF-8 STRING format and available display and conversion options can be found in the documentation for the Tc2_Utilities PLC library.

Requirements

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

5.2.1.4.1 AddBase64

This method adds a value of the data type Base64 to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 67].

Syntax

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

5.2.1.4.2 AddBool

This method adds a value of the data type BOOL to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 67].

Syntax

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

Sample call:

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

5.2.1.4.3 AddDateTime

This method adds a value of the data type DATE_AND_TIME to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 67].

Syntax

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

Sample call:

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

5.2.1.4.4 AddDcTime

This method adds a value of the data type DCTIME to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 67].

Syntax

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

Sample call:

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

5.2.1.4.5 AddDint

This method adds a value of the data type DINT to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▸ 67].

Syntax

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

Sample call:

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

5.2.1.4.6 AddFileTime

This method adds a value of the data type FILETIME to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▸ 67].

Syntax

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

Sample call:

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

5.2.1.4.7 AddHexBinary

This method adds a hex binary value to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▸ 67].

Syntax

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

Sample call:

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

5.2.1.4.8 AddKey

This method adds a new property key at the current position of the SAX writer. The value of the new property is usually set afterwards. This can be done using one of the following methods, for example: AddBase64 [▸ 66], AddBool [▸ 66], AddDateTime [▸ 66], AddDcTime [▸ 66], AddDint [▸ 67], AddFileTime [▸ 67], AddHexBinary [▸ 67], AddLint [▸ 70], AddLreal [▸ 70], AddNull [▸ 70], AddRawArray [▸ 70], AddRawObject [▸ 71], AddReal [▸ 71], AddString [▸ 71], AddUdint [▸ 72], AddUlint [▸ 72].

Syntax

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

Sample call:

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

5.2.1.4.9 AddKeyBool

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

Syntax

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

Sample call:

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

5.2.1.4.10 AddKeyDateTime

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

Syntax

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

Sample call:

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

5.2.1.4.11 AddKeyDcTime

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

Syntax

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

Sample call:

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

5.2.1.4.12 AddKeyFileTime

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

Syntax

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

```
VAR_INPUT
  value : FILETIME;
END_VAR
```

Sample call:

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

5.2.1.4.13 AddKeyLreal

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

Syntax

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

Sample call:

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

5.2.1.4.14 AddKeyNull

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

Syntax

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

Sample call:

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

5.2.1.4.15 AddKeyNumber

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

Syntax

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

Sample call:

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

5.2.1.4.16 AddKeyString

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

Syntax

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

Sample call:

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

5.2.1.4.17 AddLint

This method adds a value of the data type LINT to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▸ 67].

Syntax

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

Sample call:

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

5.2.1.4.18 AddLreal

This method adds a value of the data type LREAL to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▸ 67].

Syntax

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

Sample call:

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

5.2.1.4.19 AddNull

This method adds the value zero to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▸ 67].

Syntax

```
METHOD AddNull
```

Sample call:

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

5.2.1.4.20 AddRawArray

This method adds a valid JSON array to a given property as a value. The array to be added must be in a valid JSON format and may only be added if the SAX writer is at a correspondingly valid position, i.e. for example, directly after a preceding AddKey() [▸ 67], StartArray() [▸ 74] or as the first call after a ResetDocument() [▸ 73].

Syntax

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

Sample call:

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

5.2.1.4.21 AddRawObject

This method adds a valid JSON object to a given property as a value. The object to be added must be in a valid JSON format and may only be added if the SAX writer is at a correspondingly valid position, i.e. for example, directly after a preceding AddKey() [▶ 67], StartArray() [▶ 74] or as the first call after a ResetDocument() [▶ 73].

Syntax

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

Sample call:

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

5.2.1.4.22 AddReal

This method adds a value of the data type REAL to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 67].

Syntax

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

Sample call:

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

5.2.1.4.23 AddString

This method adds a value of the data type STRING to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 67].

Syntax

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

Sample call:

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

5.2.1.4.24 AddUdint

This method adds a value of the data type UDINT to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▸ 67].

Syntax

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

Sample call:

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

5.2.1.4.25 AddUlint

This method adds a value of the data type ULINT to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▸ 67].

Syntax

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

Sample call:

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

5.2.1.4.26 CopyDocument

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

Syntax

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

Sample call:

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

5.2.1.4.27 EndArray

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

Syntax

```
METHOD EndArray : HRESULT
```

Sample call:

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

5.2.1.4.28 EndObject

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

Syntax

```
METHOD EndObject : HRESULT
```

Sample call:

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

5.2.1.4.29 GetDocument

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

The maximum size of the string returned by the method is 255 characters. With longer strings, the method will return a NULL string. In this case the method CopyDocument [▶ 72]() must be used.

Syntax

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

Sample call:

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

5.2.1.4.30 GetDocumentLength

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

Syntax

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

Sample call:

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

5.2.1.4.31 GetMaxDecimalPlaces

Syntax

```
METHOD GetMaxDecimalPlaces : DINT
```

5.2.1.4.32 ResetDocument

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

Syntax

```
METHOD ResetDocument : HRESULT
```

Sample call:

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

5.2.1.4.33 SetMaxDecimalPlaces

Syntax

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

5.2.1.4.34 StartArray

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

Syntax

```
METHOD StartArray : HRESULT
```

Sample call:

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

5.2.1.4.35 StartObject

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

Syntax

```
METHOD StartObject : HRESULT
```

Sample call:

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

5.2.1.5 FB_JsonSaxPrettyWriter

This function block has a difference from FB_JsonSaxWriter [▶ 65]: it adds matching whitespaces for better readability of a JSON document.

Strings in UTF-8 format

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

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

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

Further information on the UTF-8 STRING format and available display and conversion options can be found in the documentation for the Tc2_Utilities PLC library.

Requirements

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

5.2.1.5.1 AddBase64

This method adds a value of the data type Base64 to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 76].

Syntax

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

5.2.1.5.2 AddBool

This method adds a value of the data type BOOL to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 76].

Syntax

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

Sample call:

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

5.2.1.5.3 AddDateTime

This method adds a value of the data type DATE_AND_TIME to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 76].

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
```

5.2.1.5.4 AddDcTime

This method adds a value of the data type DCTIME to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 76].

Syntax

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

Sample call:

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

5.2.1.5.5 AddDint

This method adds a value of the data type DINT to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 76].

Syntax

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

Sample call:

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

5.2.1.5.6 AddFileTime

This method adds a value of the data type FILETIME to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 76].

Syntax

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

Sample call:

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

5.2.1.5.7 AddHexBinary

This method adds a hex binary value to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 76].

Syntax

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

Sample call:

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

5.2.1.5.8 AddKey

This method adds a new property key at the current position of the SAX writer. The value of the new property is usually set afterwards. This can be done using one of the following methods, for example: AddBase64 [▶ 75], AddBool [▶ 75], AddDateTime [▶ 75], AddDcTime [▶ 75], AddDint [▶ 76], AddFileTime [▶ 76], AddHexBinary [▶ 76], AddLint [▶ 79], AddLreal [▶ 79], AddNull [▶ 79], AddRawArray [▶ 79], AddRawObject [▶ 80], AddReal [▶ 80], AddString [▶ 80], AddUdint [▶ 81], AddUlint [▶ 81].

Syntax

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

Sample call:

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

5.2.1.5.9 AddKeyBool

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

Syntax

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

Sample call:

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

5.2.1.5.10 AddKeyDateTime

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

Syntax

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

Sample call:

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

5.2.1.5.11 AddKeyDcTime

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

Syntax

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

Sample call:

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

5.2.1.5.12 AddKeyFileTime

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

Syntax

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

```
VAR_INPUT
  value : FILETIME;
END_VAR
```

Sample call:

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

5.2.1.5.13 AddKeyLreal

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

Syntax

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

Sample call:

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

5.2.1.5.14 AddKeyNull

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

Syntax

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

Sample call:

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

5.2.1.5.15 AddKeyNumber

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

Syntax

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

Sample call:

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

5.2.1.5.16 AddKeyString

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

Syntax

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

Sample call:

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

5.2.1.5.17 AddLint

This method adds a value of the data type LINT to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 76].

Syntax

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

Sample call:

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

5.2.1.5.18 AddLreal

This method adds a value of the data type LREAL to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 76].

Syntax

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

Sample call:

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

5.2.1.5.19 AddNull

This method adds the value zero to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 76].

Syntax

```
METHOD AddNull
```

Sample call:

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

5.2.1.5.20 AddRawArray

This method adds a valid JSON array to a given property as a value. The array to be added must be in a valid JSON format and may only be added if the SAX writer is at a correspondingly valid position, i.e. for example, directly after a preceding AddKey() [▶ 76], StartArray() [▶ 83] or as the first call after a ResetDocument() [▶ 82].

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');
```

5.2.1.5.21 AddRawObject

This method adds a valid JSON object to a given property as a value. The object to be added must be in a valid JSON format and may only be added if the SAX writer is at a correspondingly valid position, i.e. for example, directly after a preceding AddKey() [▸ 76], StartArray() [▸ 83] or as the first call after a ResetDocument() [▸ 82].

Syntax

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

Sample call:

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

5.2.1.5.22 AddReal

This method adds a value of the data type REAL to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▸ 76].

Syntax

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

Sample call:

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

5.2.1.5.23 AddString

This method adds a value of the data type STRING to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▸ 76].

Syntax

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

Sample call:

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

5.2.1.5.24 AddUdint

This method adds a value of the data type UDINT to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▸ 76].

Syntax

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

Sample call:

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

5.2.1.5.25 AddUlint

This method adds a value of the data type ULINT to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▸ 76].

Syntax

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

Sample call:

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

5.2.1.5.26 CopyDocument

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

Syntax

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

Sample call:

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

5.2.1.5.27 EndArray

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

Syntax

```
METHOD EndArray : HRESULT
```

Sample call:

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

5.2.1.5.28 EndObject

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

Syntax

```
METHOD EndObject : HRESULT
```

Sample call:

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

5.2.1.5.29 GetDocument

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

The maximum size of the string returned by the method is 255 characters. With longer strings, the method will return a NULL string. In this case the method CopyDocument [▶ 81]() must be used.

Syntax

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

Sample call:

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

5.2.1.5.30 GetDocumentLength

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

Syntax

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

Sample call:

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

5.2.1.5.31 GetMaxDecimalPlaces

Syntax

```
METHOD GetMaxDecimalPlaces : DINT
```

5.2.1.5.32 ResetDocument

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

Syntax

```
METHOD ResetDocument : HRESULT
```

Sample call:

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

5.2.1.5.33 SetFormatOptions

This method is used to customize the general formatting options for an instance of the FB_JsonSaxPrettyWriter [▶ 74]. In addition to the default configuration, there is also the possibility that no line breaks are generated within an array. The array is then output in one line despite the inserted indentations.

Syntax

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

Sample call:

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

5.2.1.5.34 SetIndent

This method is used to customize the indentation.

Syntax

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

Sample call:

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

5.2.1.5.35 SetMaxDecimalPlaces

Syntax

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

5.2.1.5.36 StartArray

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

Syntax

```
METHOD StartArray : HRESULT
```

Sample call:

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

5.2.1.5.37 StartObject

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

Syntax

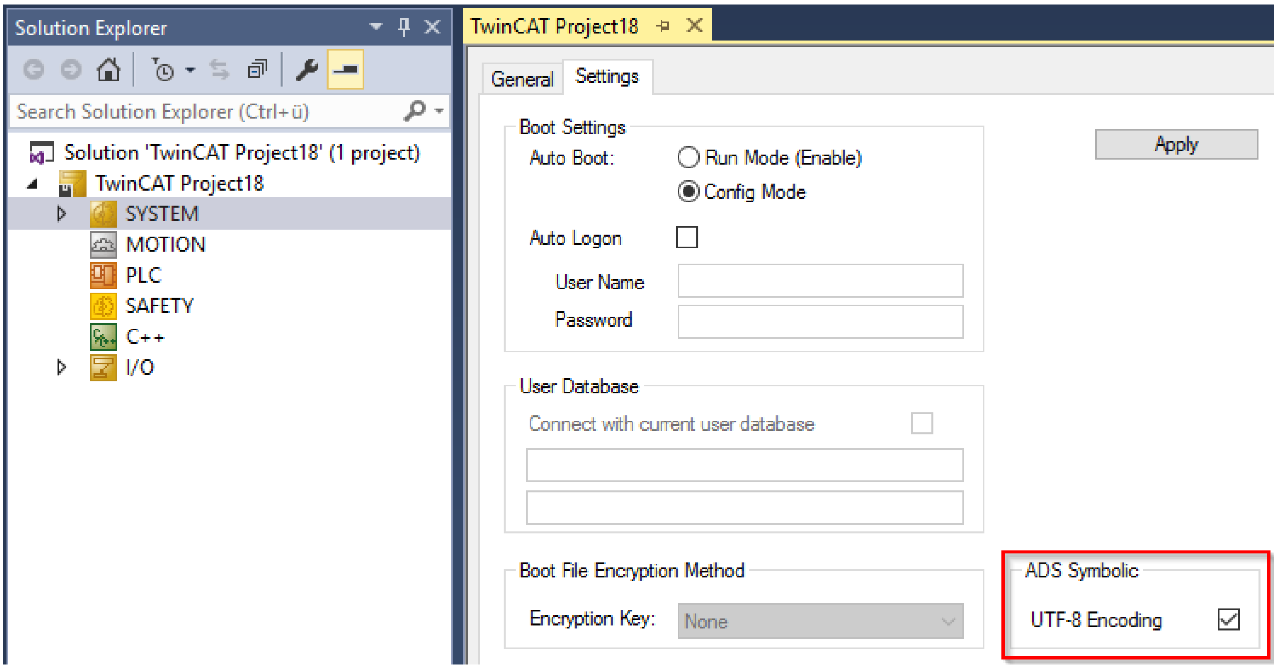
```
METHOD StartObject : HRESULT
```

Sample call:

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

5.2.1.6 FB_JsonReadWriteDataType

In order to use UTF-8 characters, e.g. in the automatic generation of metadata via the function block FB_JsonReadWriteDataType [▶ 84], the check box for the support of UTF-8 in the symbolism must be activated in the TwinCAT project. To do this, double-click on **SYSTEM** in the project tree, open the **Settings** tab and activate the corresponding check box.



Boot Settings
Auto Boot: ○ Run Mode (Enable) ● Config Mode
Auto Logon □
User Name
Password

User Database
Connect with current user database □

Boot File Encryption Method
Encryption Key: None

ADS Symbolic
UTF-8 Encoding ☑

Strings in UTF-8 format

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

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

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

Further information on the UTF-8 STRING format and available display and conversion options can be found in the documentation for the Tc2_Utilities PLC library.

Requirements

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

5.2.1.6.1 AddJsonKeyPropertiesFromSymbol

With the aid of this method, metadata can be added via PLC attributes to the JSON representation of a PLC data structure on an FB_JsonSaxWriter [▶ 65] object. The method receives as its input parameters the instance of the FB_JsonSaxWriter function block, the desired name of the JSON property that is to contain the metadata, the data type name of the structure and a string variable sProperties, which contains a list of the PLC attributes to be extracted, separated by a cross bar.

Syntax

```
METHOD AddJsonValueFromSymbol : BOOL
VAR_IN_OUT
  fbWriter    : FB_JsonSaxWriter;
END_VAR
VAR_IN_OUT CONSTANT
  sKey        : STRING;
  sDatatype   : STRING;
  sProperties : STRING;
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

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

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

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

Sample call:

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

5.2.1.6.2 AddJsonKeyValueFromSymbol

This method generates the JSON representation of a PLC data structure on an FB_JsonSaxWriter [▶ 65] 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() [▶ 86], the elements of the source structure are nested here in a JSON sub-object whose name can be specified via the input/output parameter sKey.

Syntax

```
METHOD AddJsonValueFromSymbol : BOOL
VAR_IN_OUT
  fbWriter  : FB_JsonSaxWriter;
END_VAR
VAR_IN_OUT CONSTANT
  sKey      : STRING;
  sDatatype : STRING;
END_VAR
VAR_INPUT
  nData     : UDINT;
  pData     : PVOID;
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

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

Sample call:

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

5.2.1.6.3 AddJsonValueFromSymbol

This method generates the JSON representation of a PLC data structure on an FB_JsonSaxWriter [▶ 65] object. The method receives as its input parameters the instance of the FB_JsonSaxWriter function block, the data type name of the structure, and the address and size of the source structure instance. As a result, the FB_JsonSaxWriter instance contains a valid JSON representation of the structure.

Syntax

```
METHOD AddJsonValueFromSymbol : BOOL
VAR_IN_OUT
  fbWriter : FB_JsonSaxWriter;
END_VAR
VAR_IN_OUT CONSTANT
  sDatatype : STRING;
END_VAR
VAR_INPUT
  nData : UDINT;
  pData : PVOID;
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

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

Sample call:

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

5.2.1.6.4 CopyJsonStringFromSymbol

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

Syntax

```
METHOD CopyJsonStringFromSymbol : UDINT
VAR_INPUT
  nData : UDINT;
  nDoc  : UDINT;
  pData : PVOID;
END_VAR
VAR_IN_OUT CONSTANT
  pDoc      : STRING;
  sDatatype : STRING;
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

Sample call:

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

5.2.1.6.5 CopyJsonStringFromSymbolProperties

This method generates a corresponding JSON representation of PLC attributes on a symbol. In contrast to the AddJsonKeyPropertiesFromSymbol [▶ 85] method, the result is not written to an instance of the function block FB_JsonSaxWriter, but to a string variable. The method receives as its input parameters the data type name of the symbol and a string variable that represents a list of the PLC attributes to be extracted, separated by a cross bar.

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

Syntax

```
METHOD CopyJsonStringFromSymbolProperties : UDINT
VAR_INPUT
  nDoc   : UDINT;
END_VAR
VAR_IN_OUT CONSTANT
  pDoc        : STRING;
  sDatatype   : STRING;
  sProperties : STRING;
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

Sample call:

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

5.2.1.6.6 CopySymbolNameByAddress

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

Syntax

```
METHOD CopySymbolNameByAddress : UDINT
VAR_INPUT
    nData     : UDINT;     // size of symbol
    pData     : PVOID;     // address of symbol
END_VAR
VAR_IN_OUT CONSTANT
    sName     : STRING;     // target string buffer where the symbol name should be copied to
END_VAR
VAR_INPUT
    nName     : UDINT;     // size in bytes of target string buffer
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

Sample call:

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

Requirements

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

5.2.1.6.7 GetDataTypeNameByAddress

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

Syntax

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

```
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

Sample call:

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

5.2.1.6.8 GetJsonFromSymbol

This method generates the corresponding JSON representation of a symbol. In contrast to the AddJsonValueFromSymbol() [▸ 86] method, the result is not written to an instance of the function block FB_JsonSaxWriter, but to a string variable. The method receives as its input parameters the data type name of the symbol as well as the address and size of the source symbol, e.g. of a structure instance. The address and size of the destination buffer that contains the JSON representation of the symbol after the call are transferred as further input parameters.

Syntax

```
METHOD GetJsonFromSymbol : BOOL
VAR_IN_OUT CONSTANT
  sDatatype : STRING;
END_VAR
VAR_INPUT
  nData : UDINT;
  pData : PVOID;
  nJson : REFERENCE TO UDINT;
  pJson : POINTER TO STRING;
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

Input parameter nJson

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

Sample call:

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

5.2.1.6.9 GetJsonStringFromSymbol

This method generates the corresponding JSON representation of a symbol. In contrast to the AddJsonValueFromSymbol() [▸ 86] method, the result is not written to an instance of the function block FB_JsonSaxWriter, but to a string variable. The method receives as its input parameters the data type name of the symbol as well as the address and size of the source symbol, e.g., of a structure instance.

The maximum size of the string returned by the method is 255 characters. With longer strings, the method will return a NULL string. In this case the method CopyJsonStringFromSymbol [▸ 86]() must be used.

Syntax

```
METHOD GetJsonStringFromSymbol : STRING(255)
VAR_IN_OUT CONSTANT
  sDatatype : STRING;
END_VAR
VAR_INPUT
  nData : UDINT;
  pData : PVOID;
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

Sample call:

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

5.2.1.6.10 GetJsonStringFromSymbolProperties

This method generates a corresponding JSON representation of PLC attributes on a symbol. In contrast to the AddJsonKeyPropertiesFromSymbol [▶ 85] method, the result is not written to an instance of the function block FB_JsonSaxWriter, but to a string variable. The method receives as its input parameters the data type name of the symbol and a string variable that represents a list of the PLC attributes to be extracted, separated by a cross bar. The result is returned directly as the return value of the method.

The maximum size of the string returned by the method is 255 characters. With longer strings, the method will return a NULL string. In this case the method CopyJsonStringFromSymbolProperties [▶ 86]() must be used.

Syntax

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

Sample call:

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

5.2.1.6.11 GetSizeJsonStringFromSymbol

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

Syntax

```
METHOD GetSizeJsonStringFromSymbol : UDINT
VAR_INPUT
  nData   :UDINT;
  pData   :PVOID;
END_VAR
VAR_IN_OUT CONSTANT
  sDatatype    : STRING;
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

Sample call:

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

5.2.1.6.12 GetSizeJsonStringFromSymbolProperties

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

Syntax

```
METHOD GetSizeJsonStringFromSymbolProperties : UDINT
VAR_IN_OUT CONSTANT
  sDatatype    : STRING;
  sProperties : STRING;
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

Sample call:

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

5.2.1.6.13 GetSymbolNameByAddress

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

The maximum size of the string returned by the method is 255 characters. With longer strings, the method will return a null string. In this case the method CopySymbolNameByAddress() [▶ 87] must be used.

Syntax

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

Sample call:

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

5.2.1.6.14 SetSymbolFromJson

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

Syntax

```
METHOD SetSymbolFromJson : BOOL
VAR_IN_OUT CONSTANT
  sJson : STRING;
  sDatatype : STRING;
END_VAR
VAR_INPUT
  nData : UDINT;
  pData : PVOID;
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

Sample call:

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

5.2.1.7 FB_XmlDomParser

Strings in UTF-8 format

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

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

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

Further information on the UTF-8 STRING format and available display and conversion options can be found in the documentation for the Tc2_Utilities PLC library.

Requirements

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

5.2.1.7.1 AppendAttribute

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

Syntax

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

Sample call:

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

5.2.1.7.2 AppendAttributeAsBool

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

Syntax

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

Sample call:

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

5.2.1.7.3 AppendAttributeAsDouble

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

Syntax

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

Sample call:

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

5.2.1.7.4 AppendAttributeAsFloat

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

Syntax

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

Sample call:

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

5.2.1.7.5 AppendAttributeAsInt

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

Syntax

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

Sample call:

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

5.2.1.7.6 AppendAttributeAsLint

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

Syntax

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

Sample call:

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

5.2.1.7.7 AppendAttributeAsUint

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

Syntax

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

Sample call:

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

5.2.1.7.8 AppendAttributeAsUlint

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

Syntax

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

Sample call:

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

5.2.1.7.9 AppendAttributeCopy

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

Syntax

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

Sample call:

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

5.2.1.7.10 AppendChild

This method inserts a new node below an existing node. The value of the new node 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);
```

5.2.1.7.11 AppendChildAsBool

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

Syntax

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

Sample call:

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

5.2.1.7.12 AppendChildAsDouble

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

Syntax

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

Sample call:

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

5.2.1.7.13 AppendChildAsFloat

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

Syntax

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

Sample call:

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

5.2.1.7.14 AppendChildAsInt

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

Syntax

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

Sample call:

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

5.2.1.7.15 AppendChildAsLint

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

Syntax

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

Sample call:

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

5.2.1.7.16 AppendChildAsUint

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

Syntax

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

Sample call:

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

5.2.1.7.17 AppendChildAsUlint

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

Syntax

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

Sample call:

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

5.2.1.7.18 AppendCopy

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

Syntax

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

Sample call:

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

5.2.1.7.19 AppendNode

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

Syntax

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

Sample call:

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

5.2.1.7.20 Attributes

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

Syntax

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

Sample call:

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

5.2.1.7.21 AttributeAsBool

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

Syntax

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

Sample call:

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

5.2.1.7.22 AttributeAsDouble

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

Syntax

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

Sample call:

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

5.2.1.7.23 AttributeAsFloat

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

Syntax

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

Sample call:

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

5.2.1.7.24 AttributeAsInt

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

Syntax

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

Sample call:

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

5.2.1.7.25 AttributeAsLint

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

Syntax

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

Sample call:

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

5.2.1.7.26 AttributeAsUint

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

Syntax

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

Sample call:

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

5.2.1.7.27 AttributeAsUlint

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

Syntax

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

Sample call:

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

5.2.1.7.28 AttributeBegin

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

Syntax

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

Sample call:

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

5.2.1.7.29 AttributeFromIterator

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

Syntax

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

Sample call:

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

5.2.1.7.30 AttributeName

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

Syntax

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

Sample call:

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

5.2.1.7.31 Attributes

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

Syntax

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

Sample call:

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

5.2.1.7.32 AttributeText

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

Syntax

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

Sample call:

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

5.2.1.7.33 Begin

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

Syntax

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

Sample call:

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

5.2.1.7.34 BeginByName

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

Syntax

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

Sample call:

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

5.2.1.7.35 Child

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

Syntax

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

Sample call:

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

5.2.1.7.36 ChildByAttribute

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

Syntax

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

Sample call:

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

5.2.1.7.37 ChildByAttributeAndName

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

Syntax

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

Sample call:

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

5.2.1.7.38 ChildByName

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

Syntax

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

Sample call:

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

5.2.1.7.39 Children

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

Syntax

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

Sample call:

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

5.2.1.7.40 ChildrenByName

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

Syntax

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

Sample call:

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

5.2.1.7.41 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);
```

5.2.1.7.42 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));
```

5.2.1.7.43 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));
```

5.2.1.7.44 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));
```

5.2.1.7.45 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));
```

5.2.1.7.46 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');
```

5.2.1.7.47 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);
```

5.2.1.7.48 GetDocumentLength

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

Syntax

```
METHOD GetDocumentLength : UDINT
```

Sample call:

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

5.2.1.7.49 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();
```

5.2.1.7.50 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);
```

5.2.1.7.51 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);
```

5.2.1.7.52 GetRootNode

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

Syntax

```
METHOD GetRootNode : SXmlNode
```

Sample call:

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

5.2.1.7.53 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);
```

5.2.1.7.54 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');
```

5.2.1.7.55 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');
```

5.2.1.7.56 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);
```

5.2.1.7.57 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
```

5.2.1.7.58 LoadDocumentFromFile

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

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

Syntax

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

Sample call:

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

5.2.1.7.59 NewDocument

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

Syntax

```
METHOD NewDocument : BOOL
```

Sample call:

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

5.2.1.7.60 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
```

5.2.1.7.61 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);
```

5.2.1.7.62 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
```

5.2.1.7.63 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);
```

5.2.1.7.64 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.NextSiblingByName(xmlNode, 'SomeName');
```

5.2.1.7.65 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
```

5.2.1.7.66 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);
```

5.2.1.7.67 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);
```

5.2.1.7.68 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);
```

5.2.1.7.69 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);
```

5.2.1.7.70 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);
```

5.2.1.7.71 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);
```

5.2.1.7.72 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);
```

5.2.1.7.73 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);
```

5.2.1.7.74 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);
```

5.2.1.7.75 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);
```

5.2.1.7.76 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);
```

5.2.1.7.77 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');
```

5.2.1.7.78 SaveDocumentToFile

This method saves the current XML document in a file. The absolute path to the file is transferred to the method as input parameter.

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

Syntax

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

Sample call:

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

5.2.1.7.79 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');
```

5.2.1.7.80 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);
```

5.2.1.7.81 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);
```

5.2.1.7.82 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);
```

5.2.1.7.83 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);
```

5.2.1.7.84 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);
```

5.2.1.7.85 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);
```

5.2.1.7.86 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);
```

5.2.1.7.87 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);
```

5.2.1.7.88 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);
```

5.2.1.7.89 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);
```

5.2.1.7.90 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);
```

5.2.1.7.91 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);
```

5.2.1.7.92 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);
```

5.2.1.7.93 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);
```

5.2.1.7.94 SetChildAsUlint

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

Syntax

```
METHOD SetChildAsUlint : SXmlNode
VAR_INPUT
  n : SXmlNode;
  value : ULINT;
END_VAR
```

Sample call:

```
xmlNode := fbXml.SetChildAsUlint(xmlExistingNode, 42);
```

5.2.1.8 FB_JwtEncode

The function block enables the creation and signing of a JSON Web Token (JWT).

Syntax

Definition:

```
FUNCTION_BLOCK FB_JwtEncode
VAR_INPUT
  bExecute       : BOOL;
  sHeaderAlg     : STRING(46);
  sPayload       : STRING(1023);
  sKeyFilePath   : STRING(511);
  tTimeout       : TIME;
  pKey           : PVOID;
  nKeySize       : UDINT;
  nJwtSize       : UDINT;
```

```
END_VAR
VAR_IN_OUT CONSTANT
  sJwt            : STRING;
END_VAR
VAR_OUTPUT
  bBusy           : BOOL;
  bError          : BOOL;
  hrErrorCode     : HRESULT;
  initStatus      : HRESULT;
END_VAR
```

Inputs

Name	Type	Description
bExecute	BOOL	A rising edge activates processing of the function block.
sHeaderAlg	STRING(46)	The algorithm to be used for the JWT header, e.g. RS256.
sPayload	STRING(1023)	The JWT payload to be used.
sKeyFilePath	STRING(511)	Path to the private key to be used for the signature of the JWT.
tTimeout	TIME	ADS timeout, which is used internally for file access to the private key.
pKey	PVOID	Buffer for the private key to be read.
nKeySize	UDINT	Maximum size of the buffer.
sJwt	STRING	Contains the fully coded and signed JWT after the function block has been processed.
nJwtSize	UDINT	Size of the generated JWT including zero termination.

Outputs

Name	Type	Description
bBusy	BOOL	Is TRUE as long as processing of the function block is in progress.
bError	BOOL	Becomes TRUE as soon as an error situation occurs.
hrErrorCode	HRESULT	Returns an error code if the `bError` output is set. An explanation of the possible error codes can be found in the Appendix.
initStatus	HRESULT	Returns an error code in case of a failed initialization of the function block.

Requirements

TwinCAT version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4024.4	x86, x64, ARM	Tc3_JsonXml 3.3.6.0

5.2.2 Interfaces

5.2.2.1 ITcJsonSaxHandler

5.2.2.1.1 OnBool

This callback method is triggered if a value of the data type BOOL was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnBool : HRESULT
VAR_INPUT
    value : BOOL;
END_VAR
```

5.2.2.1.2 OnDint

This callback method is triggered if a value of the data type DINT was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnDint : HRESULT
VAR_INPUT
    value : DINT;
END_VAR
```

5.2.2.1.3 OnEndArray

This callback method is triggered if a square closing bracket, which corresponds to the JSON synonym for an ending array, was found at the position of the SAX reader. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnEndArray : HRESULT
```

5.2.2.1.4 OnEndObject

This callback method is triggered if a curly closing bracket, which corresponds to the JSON synonym for an ending object, was found at the position of the SAX reader. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnEndObject : HRESULT
```

5.2.2.1.5 OnKey

This callback method is triggered if a property was found at the position of the SAX reader. The property name lies on the input/output parameter key and its length on the input parameter len. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnKey : HRESULT
VAR_IN_OUT CONSTANT
    key : STRING;
END_VAR
VAR_INPUT
    len : UDINT;
END_VAR
```

5.2.2.1.6 OnLint

This callback method is triggered if a value of the data type LINT was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnLint : HRESULT
VAR_INPUT
    value : LINT;
END_VAR
```

5.2.2.1.7 OnLreal

This callback method is triggered if a value of the data type LREAL was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnLreal : HRESULT
VAR_INPUT
    value : LREAL;
END_VAR
```

5.2.2.1.8 OnNull

This callback method is triggered if a NULL value was found at the position of the SAX reader. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnNull : HRESULT
```

5.2.2.1.9 OnStartArray

This callback method is triggered if a square opening bracket, which corresponds to the JSON synonym for a starting array, was found at the position of the SAX reader. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnStartArray : HRESULT
```

5.2.2.1.10 OnStartObject

This callback method is triggered if a curly opening bracket, which corresponds to the JSON synonym for a starting object, was found at the position of the SAX reader. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnStartObject : HRESULT
```

5.2.2.1.11 OnString

This callback method is triggered if a value of the data type STRING was found at the position of the SAX reader. The In/Out parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnString : HRESULT
VAR_IN_OUT CONSTANT
    value : STRING;
END_VAR
```

```
VAR_INPUT
    len : UDINT;
END_VAR
```

5.2.2.1.12 OnUdint

This callback method is triggered if a value of the data type UDINT was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnUdint : HRESULT
VAR_INPUT
    value : UDINT;
END_VAR
```

5.2.2.1.13 OnUlint

This callback method is triggered if a value of the data type ULINT was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnUlint : HRESULT
VAR_INPUT
    value : ULINT;
END_VAR
```

5.2.2.2 ITcJsonSaxValues

5.2.2.2.1 OnBoolValue

This callback method is triggered if a value of the data type BOOL was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnBoolValue : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
    value : BOOL;
END_VAR
```

5.2.2.2.2 OnDintValue

This callback method is triggered if a value of the data type DINT was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnDintValue : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
    value : DINT;
END_VAR
```

5.2.2.2.3 OnLintValue

This callback method is triggered if a value of the data type LINT was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnLintValue : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
    value : LINT;
END_VAR
```

5.2.2.2.4 OnLrealValue

This callback method is triggered if a value of the data type LREAL was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnLrealValue : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
    value : LREAL;
END_VAR
```

5.2.2.2.5 OnNullValue

This callback method is triggered if a NULL value was found at the position of the SAX reader. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnNull : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
END_VAR
```

5.2.2.2.6 OnStringValue

This callback method is triggered if a value of the data type STRING was found at the position of the SAX reader. The input/output parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnStringValue : HRESULT
VAR_IN_OUT CONSTANT
    value : STRING;
END_VAR
VAR_INPUT
    len : UDINT;
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
END_VAR
```

5.2.2.2.7 OnUdintValue

This callback method is triggered if a value of the data type UDINT was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnUdintValue : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
    value : UDINT;
END_VAR
```

5.2.2.2.8 OnUlintValue

This callback method is triggered if a value of the data type ULINT was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

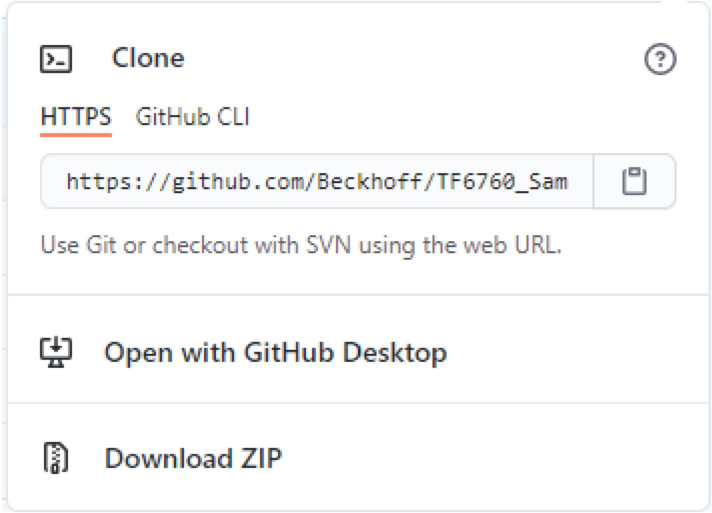
```
METHOD OnUlintValue : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
    value : ULINT;
END_VAR
```

6 Samples

6.1 IotHttpSamples

Each sample is provided as a separate PLC project. Samples are available for AWS IoT Core, OpenWeatherMap, Philips Hue, Postman Echo,Telegram Messenger and for AWS service configuration.

Sample code and configurations for this product can be obtained from the corresponding repository on GitHub: https://github.com/Beckhoff/TF6760_Samples. There you have the option to clone the repository or download a ZIP file containing the sample.

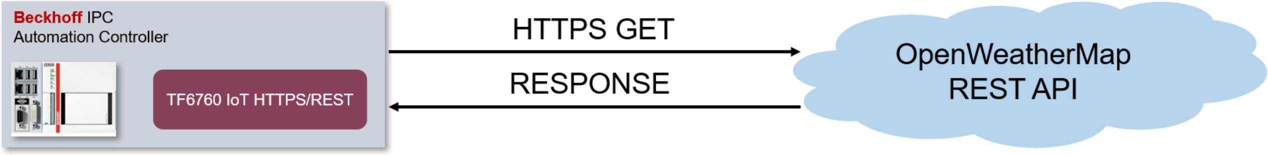

Clone
HTTPS
GitHub CLI
https://github.com/Beckhoff/TF6760_Sam
Use Git or checkout with SVN using the web URL.
Open with GitHub Desktop
Download ZIP

PLC projects are divided into a MAIN part and various function blocks. HTTP command-specific functions (GET, POST, PUT) are encapsulated in the named function blocks, in order to keep the program uncluttered.

The MAIN part of each sample is used to configure the HTTP client function blocks. Once the client function blocks are configured, the various HTTP commands can be triggered from the MAIN part. The instantiated HTTP client function blocks are then passed through the various encapsulated function blocks.

6.1.1 OpenWeatherMap

This sample illustrates the communication with an HTTPS/REST API. HTTPS GET requests are sent to obtain weather data from the OpenWeatherMap REST API. The response of the web service containing the data is in a JSON format.



To send requests to the OpenWeatherMap REST API, the user must create an account for OpenWeatherMap to receive an application ID. In combination with a location identifier (e.g. latitude/longitude, city ID, city name), the application ID can be used to query weather data.

API documentation for various use cases (16-day forecast, current data etc.) can be found here: https://openweathermap.org/api.

```
PROGRAM MAIN
VAR
    // trigger command execution for OpenWeatherMap samples
    bGetOpenWeatherMap               : BOOL;

    fbHttpClientOpenWeatherMap      : FB_IotHttpClient :=(sHostName:='api.openweathermap.org',
                                     bKeepAlive:=TRUE, tConnectionTimeout:=T#10S);

    fbHttpGetOpenWeatherMap        : FB_TestHTTP_Get_openWeatherMap;
END_VAR
```

```
//init client parameters at startup
IF NOT fbHttpClientOpenWeatherMap.bConfigured THEN
    fbHttpClientOpenWeatherMap.nHostPort:= 443;
    fbHttpClientOpenWeatherMap.stTLS.bNoServerCertCheck:= TRUE;
END_IF

IF fbHttpClientOpenWeatherMap.bConfigured THEN
    fbHttpGetOpenWeatherMap(bSend:=bGetOpenWeatherMap, fbClient:=fbHttpClientOpenWeatherMap);
END_IF

fbHttpClientOpenWeatherMap.Execute();
```

6.1.1.1 Get

1. The function block is started by writing the variable bGetOpenWeatherMap in the main program to TRUE.
2. The rising edge is then used to send a GET request with the IotHttpRequest function block.
3. After the request has been finished, the error handling is processed. When neither the function block itself nor the HTTP status code display an error, the JSON response from the webserver is parsed.

```
FUNCTION_BLOCK FB_TestHTTP_Get_openWeatherMap
VAR_INPUT
    bSend              : BOOL;
END_VAR
VAR_IN_OUT
    fbClient           : FB_IotHttpClient;
END_VAR
VAR_OUTPUT
    bBusy              : BOOL;
    bError             : BOOL;
END_VAR
VAR
    fbRequest          : FB_IotHttpRequest;
    fbJson             : FB_JsonDomParser;
    nState             : UDINT;
    RisingEdge         : R_TRIG;

    bGetContentResult : BOOL;
    sContent           : STRING(511);

    bGetJsonResult     : BOOL;
    jsonDoc            : SJsonValue;
    jsonVal            : SJsonValue;
    sResultValue       : STRING;

    nReqCount          : UDINT;
    nResCount          : UDINT;
    nValidResCount     : UDINT;
    nErrCount          : UDINT;
END_VAR
```

```
RisingEdge(CLK:= bSend);
CASE nState OF
0:
    IF RisingEdge.Q THEN
        IF fbRequest.SendRequest(sUri:= '/data/2.5/weather?id=123456&APPID=123456abcdef',
                                 fbClient:= fbClient,
                                 eRequestType:= ETcIotHttpRequestType.HTTP_Get, 0, 0, 0) THEN
            nState:= 1;
            nReqCount:= nReqCount+1;
            bBusy:= TRUE;
            bError:= FALSE;
        END_IF
    END_IF
1:
    IF NOT fbRequest.bBusy THEN
        bError:= TRUE;
        IF NOT fbRequest.bError THEN
            bGetContentResult:= fbRequest.GetContent(pContent:= ADR(sContent),
                                                     nContentSize:= SIZEOF(sContent),
                                                     bSetNullTermination:= TRUE);
            IF fbRequest.nStatusCode >= 200 AND fbRequest.nStatusCode < 300 THEN
```

```
                bGetJsonResult:= FALSE;
                jsonDoc:= fbRequest.GetJsonDomContent(fbJson);
                IF jsonDoc <> 0 THEN
                    ; //do something with the weather data
                    nValidResCount:= nValidResCount+1;
                    bError:= FALSE;
                END_IF
                nResCount:= nResCount+1;
            END_IF
        END_IF
        nState:= 0;
        bBusy:= FALSE;
        IF bError THEN
            nErrCount:= nErrCount+1;
        END_IF
    END_IF
END_CASE
```

6.1.2 PostmanEcho

This part of the sample includes three HTTP commands (GET, POST, PUT) to communicate with Postman Echo, which is a testing service for REST clients. Additionally, there is also an example for a GET request with header authentification.

The Postman Echo API is especially for developers in order to test HTTP functionality and delivers even many more possibilities as are shown in our sample. For example different authentication methods can be tested with Postman Echo.

```
PROGRAM MAIN
VAR
    // trigger command execution for Postman-Echo samples
    bGet, bPost, bPut, bHeaderAuth : BOOL;

    fbHttpClientPostman            : FB_IotHttpClient :=(sHostName:='postman-echo.com',
                                     bKeepAlive:=TRUE, tConnectionTimeout:=T#10S);

    fbHttpGet                      : FB_TestHTTP_Get;
    fbHttpPost                     : FB_TestHTTP_Post;
    fbHttpPut                      : FB_TestHTTP_Put;
    fbHttpHeaderAuth               : FB_TestHTTP_HeaderAuth;
END_VAR
```

```
//init client parameters at startup
IF NOT fbHttpClientPostman.bConfigured THEN
    fbHttpClientPostman.nHostPort:= 80;
END_IF

IF fbHttpClientPostman.bConfigured THEN
    fbHttpGet(bSend:=bGet, fbClient:=fbHttpClientPostman);
    fbHttpPost(bSend:=bPost, fbClient:=fbHttpClientPostman);
    fbHttpPut(bSend:=bPut, fbClient:=fbHttpClientPostman);
    fbHttpHeaderAuth(bSend:=bHeaderAuth, fbClient:=fbHttpClientPostman);
END_IF

fbHttpClientPostman.Execute();
```

6.1.2.1 Get

1. The function block is started by writing the variable bGet in the main program to TRUE.
2. The rising edge is then used to send a GET request with the IotHttpRequest function block.
3. After the request has been finished, the error handling is processed. When neither the function block itself nor the HTTP status code display an error, the JSON response from the webserver is parsed.

```
FUNCTION_BLOCK FB_TestHTTP_Get
VAR_INPUT
    bSend               : BOOL;
END_VAR
VAR_IN_OUT
    fbClient            : FB_IotHttpClient;
END_VAR
VAR_OUTPUT
    bBusy               : BOOL;
    bError              : BOOL;
END_VAR
VAR
    fbRequest           : FB_IotHttpRequest;
    fbJson              : FB_JsonDomParser;
    nState              : UDINT;
    RisingEdge          : R_TRIG;

    bGetContentResult   : BOOL;
    sContent            : STRING(511);

    bGetJsonResult      : BOOL;
    jsonDoc             : SJsonValue;
    jsonVal             : SJsonValue;
    sResultValue        : STRING;

    nReqCount           : UDINT;
    nResCount           : UDINT;
    nValidResCount      : UDINT;
    nErrCount           : UDINT;
END_VAR
```

```
RisingEdge(CLK:= bSend);
CASE nState OF
0:
    IF RisingEdge.Q THEN
        IF fbRequest.SendRequest(sUri:= '/get?foo1=bar1&foo2=bar2',
                                 fbClient:= fbClient,
                                 eRequestType:= ETcIotHttpRequestType.HTTP_GET, 0, 0, 0) THEN
            nState:= 1;
            nReqCount:= nReqCount+1;
            bBusy:= TRUE;
            bError:= FALSE;
        END_IF
    END_IF
1:
    IF NOT fbRequest.bBusy THEN
        bError:= TRUE;
        IF NOT fbRequest.bError THEN
            bGetContentResult:= fbRequest.GetContent(pContent:= ADR(sContent),
                                                     nContentSize:= SIZEOF(sContent),
                                                     bSetNullTermination:= TRUE);
            IF fbRequest.nStatusCode >= 200 AND fbRequest.nStatusCode < 300 THEN
                bGetJsonResult:= FALSE;
                jsonDoc:= fbRequest.GetJsonDomContent(fbJson);
                IF jsonDoc <>0 THEN
                    bGetJsonResult:= TRUE;
                    IF fbJson.HasMember(jsonDoc, 'args') THEN
                        jsonVal:= fbJson.FindMember(jsonDoc, 'args');
                        IF fbJson.HasMember(jsonVal, 'foo2') THEN
                            jsonVal:= fbJson.FindMember(jsonVal, 'foo2');
                            nValidResCount:= nValidResCount+1;
                            bError:= FALSE;
                            IF fbJson.IsString(jsonVal) THEN
                                sResultValue:= fbJson.GetString(jsonVal);
                            END_IF
                        END_IF
                    END_IF
                END_IF
                nResCount:= nResCount+1;
            END_IF
        END_IF
        nState:= 0;
        bBusy:= FALSE;
        IF bError THEN
            nErrCount:= nErrCount+1;
        END_IF
    END_IF
END_CASE
```

6.1.2.2 Post

1. The function block is started by writing the variable bPost in the main program to TRUE.
2. The rising edge is then used to send a POST request with the IotHttpRequest function block.

3. After the request has been finished, the error handling is processed. When neither the function block itself nor the HTTP status code display an error, the JSON response from the webserver is parsed. The JSON response contains the nReqCount variable that has been send to the webserver.

```
FUNCTION_BLOCK FB_TestHTTP_Post
VAR_INPUT
    bSend               : BOOL;
END_VAR
VAR_IN_OUT
    fbClient            : FB_IotHttpClient;
END_VAR
VAR_OUTPUT
    bBusy               : BOOL;
    bError              : BOOL;
END_VAR
VAR
    fbRequest           : FB_IotHttpRequest;
    fbJson              : FB_JsonDomParser;
    nState              : UDINT;
    RisingEdge          : R_TRIG;

    bGetContentResult   : BOOL;
    sContent            : STRING(511);

    bGetJsonResult      : BOOL;
    jsonDoc             : SJsonValue;
    jsonVal             : SJsonValue;
    sResultValue        : STRING;

    nReqCount           : UDINT;
    nResCount           : UDINT;
    nValidResCount      : UDINT;
    nErrCount           : UDINT;
END_VAR
```

```
RisingEdge(CLK:= bSend);
CASE nState OF
0:
    IF RisingEdge.Q THEN
        sContent:= UDINT_TO_STRING(nReqCount+1);
        IF fbRequest.SendRequest(sUri:= '/post?hello=world', fbClient:= fbClient,
                                 eRequestType:= ETcIotHttpRequestType.HTTP_POST,
                                 pContent:= ADR(sContent),
                                 nContentSize:= LEN2(ADR(sContent)), 0) THEN
            nState:= 1;
            nReqCount:= nReqCount+1;
            bBusy:= TRUE;
            bError:= FALSE;
        END_IF
    END_IF
1:
    IF NOT fbRequest.bBusy THEN
        bError:= TRUE;
        IF NOT fbRequest.bError THEN
            bGetContentResult:= fbRequest.GetContent(pContent:= ADR(sContent),
                                                     nContentSize:= SIZEOF(sContent),
                                                     bSetNullTermination:= TRUE);
            IF fbRequest.nStatusCode >= 200 AND fbRequest.nStatusCode < 300 THEN
                bGetJsonResult:= FALSE;
                jsonDoc:= fbRequest.GetJsonDomContent(fbJson);
                IF jsonDoc <>0 THEN
                    bGetJsonResult:= TRUE;
                    IF fbJson.HasMember(jsonDoc, 'data') THEN
                        jsonVal:= fbJson.FindMember(jsonDoc, 'data');
                        sResultValue:= fbJson.GetString(jsonVal);
                        IF STRING_TO_UDINT(sResultValue)= nReqCount THEN
                            nValidResCount:= nValidResCount+1;
                            bError:= FALSE;
                        END_IF
                    END_IF
                END_IF
                nResCount:= nResCount+1;
            END_IF
        END_IF
        nState:= 0;
        bBusy:= FALSE;
        IF bError THEN
            nErrCount:= nErrCount+1;
        END_IF
    END_IF
END_CASE
```

6.1.2.3 Put

1. The function block is started by writing the variable bPut in the main program to TRUE.
2. The rising edge is then used to send a PUT request with the IotHttpRequest function block.
3. After the request has been finished, the error handling is processed. When neither the function block itself nor the HTTP status code display an error, the JSON response from the webserver is parsed. The JSON response contains the nReqCount variable that has been send to the webserver.

```
FUNCTION_BLOCK FB_TestHTTP_Put
VAR_INPUT
    bSend               : BOOL;
END_VAR
VAR_IN_OUT
    fbClient            : FB_IotHttpClient;
END_VAR
VAR_OUTPUT
    bBusy               : BOOL;
    bError              : BOOL;
END_VAR
VAR
    fbRequest           : FB_IotHttpRequest;
    fbJson              : FB_JsonDomParser;
    nState              : UDINT;
    RisingEdge          : R_TRIG;

    bGetContentResult   : BOOL;
    sContent            : STRING(511);

    bGetJsonResult      : BOOL;
    jsonDoc             : SJsonValue;
    jsonVal             : SJsonValue;
    sResultValue        : STRING;

    nReqCount           : UDINT;
    nResCount           : UDINT;
    nValidResCount      : UDINT;
    nErrCount           : UDINT;
END_VAR
```

```
RisingEdge(CLK:= bSend);
CASE nState OF
0:
    IF RisingEdge.Q THEN
        sContent:= UDINT_TO_STRING(nReqCount+1);
        IF fbRequest.SendRequest(sUri:= '/put', fbClient:= fbClient,
                                 eRequestType:= ETcIotHttpRequestType.HTTP_PUT,
                                 pContent:= ADR(sContent),
                                 nContentSize:= LEN2(ADR(sContent)), 0) THEN
            nState:= 1;
            nReqCount:= nReqCount+1;
            bBusy:= TRUE;
            bError:= FALSE;
        END_IF
    END_IF
1:
    IF NOT fbRequest.bBusy THEN
        bError:= TRUE;
        IF NOT fbRequest.bError THEN
            bGetContentResult:= fbRequest.GetContent(pContent:= ADR(sContent),
                                                     nContentSize:= SIZEOF(sContent),
                                                     bSetNullTermination:= TRUE);
            IF fbRequest.nStatusCode >= 200 AND fbRequest.nStatusCode < 300 THEN
                bGetJsonResult:= FALSE;
                jsonDoc:= fbRequest.GetJsonDomContent(fbJson);
                IF jsonDoc <>0 THEN
                    bGetJsonResult:= TRUE;
                    IF fbJson.HasMember(jsonDoc, 'data') THEN
                        jsonVal:= fbJson.FindMember(jsonDoc, 'data');
                        sResultValue:= fbJson.GetString(jsonVal);
                        IF STRING_TO_UDINT(sResultValue)= nReqCount THEN
                            nValidResCount:= nValidResCount+1;
                            bError:= FALSE;
                        END_IF
                    END_IF
                END_IF
                nResCount:= nResCount+1;
            END_IF
        END_IF
        nState:= 0;
        bBusy:= FALSE;
        IF bError THEN
            nErrCount:= nErrCount+1;
        END_IF
    END_IF
END_CASE
```

6.1.2.4 Header Authentication

1. The function block is started by writing the variable bHeaderAuth in the main program to TRUE.
2. The rising edge is then used to send a GET request with the IotHttpRequest function block. This request contains an additional header field for authentication.
3. After the request has been finished, the error handling is processed. When neither the function block itself nor the HTTP status code display an error, the JSON response from the webserver is parsed. The JSON response contains the information if the authentication has been successful.

```
FUNCTION_BLOCK FB_TestHTTP_HeaderAuth
VAR_INPUT
    bSend                : BOOL;
END_VAR
VAR_IN_OUT
    fbClient             : FB_IotHttpClient;
END_VAR
VAR_OUTPUT
    bBusy                : BOOL;
    bError               : BOOL;
END_VAR
VAR
    fbRequest            : FB_IotHttpRequest;
    fbHeader             : FB_IotHttpHeaderFieldMap;
    fbJson               : FB_JsonDomParser;
    nState               : UDINT;
    RisingEdge           : R_TRIG;

    bGetContentResult    : BOOL;
    sContent             : STRING(511);

    bGetJsonResult       : BOOL;
    jsonDoc              : SJsonValue;
    jsonVal              : SJsonValue;
    bResultValue         : BOOL;

    nReqCount            : UDINT;
    nResCount            : UDINT;
    nValidResCount       : UDINT;
    nErrCount            : UDINT;
    bOnce                : BOOL:= TRUE;
END_VAR
```

```
IF bOnce THEN
    fbHeader.AddField('Authorization', 'Basic cG9zdG1hbjpwYXNzd29yZA==', FALSE);
    bOnce:= FALSE;
END_IF

RisingEdge(CLK:= bSend);
CASE nState OF
0:
    IF RisingEdge.Q THEN
        IF fbRequest.SendRequest(sUri:= '/basic-auth', fbClient:= fbClient,
                                 eRequestType:= ETcIotHttpRequestType.HTTP_GET, 0, 0, fbHeader) THEN
            nState:= 1;
            nReqCount:= nReqCount+1;
            bBusy:= TRUE;
            bError:= FALSE;
        END_IF
    END_IF
1:
    IF NOT fbRequest.bBusy THEN
```

```
        bError:= TRUE;
        IF NOT fbRequest.bError THEN
            bGetContentResult:= fbRequest.GetContent(pContent:= ADR(sContent),
                                                     nContentSize:= SIZEOF(sContent),
                                                     bSetNullTermination:= TRUE);
            IF fbRequest.nStatusCode >= 200 AND fbRequest.nStatusCode < 300 THEN
                bGetJsonResult:= FALSE;
                jsonDoc:= fbRequest.GetJsonDomContent(fbJson);
                IF jsonDoc <>0 THEN
                    bGetJsonResult:= TRUE;
                    IF fbJson.HasMember(jsonDoc, 'authenticated') THEN
                        jsonVal:= fbJson.FindMember(jsonDoc, 'authenticated');
                        IF fbJson.IsBool(jsonVal) THEN
                            bResultValue:= fbJson.GetBool(jsonVal);
                            nValidResCount:= nValidResCount+1;
                            bError:= FALSE;
                        END_IF
                    END_IF
                END_IF
                nResCount:= nResCount+1;
            END_IF
        END_IF
        nState:= 0;
        bBusy:= FALSE;
        IF bError THEN
            nErrCount:= nErrCount+1;
        END_IF
    END_IF
END_CASE
```

6.1.3 AWS IoT

This sample shows two different scenarios with AWS IoT. A GET request on an IoT Shadow and a POST request on the AWS IoT Core message broker are shown.

The GET request is similar to the “OpenWeatherMap” sample, only the Uri has been changed to the Uri of Shadow of the IoT Thing, regarding to the AWS REST API.

The user is required to have an AWS account in order to have an IoT Core endpoint and the possibility to create certificates for the communication with TwinCAT.

```
PROGRAM MAIN
VAR
    // trigger command execution for AWS IoT Core samples
    bGetAwsIotShadow, bPostAwsIot  : BOOL;

    fbHttpClientAwsIot             : FB_IotHttpClient :=(sHostName:='youradress.amazonaws.com',
                                     bKeepAlive:=FALSE, tConnectionTimeout:=T#10S);

    fbHttpGetAwsIotShadow          : FB_TestHTTP_Get_awsIotShadow;
    fbHttpPostAwsIot               : FB_TestHTTP_Post_awsIot;
END_VAR
```

```
//init client parameters at startup
IF NOT fbHttpClientAwsIot.bConfigured THEN
    fbHttpClientAwsIot.stTLS.sCA:= 'C:\TwinCAT\3.1\Config\Certificates\AWS\AmazonRootCA1.pem';
    fbHttpClientAwsIot.stTLS.sCert:= 'C:\TwinCAT\3.1\Config\Certificates\AWS\certificate.pem.crt';
    fbHttpClientAwsIot.stTLS.sKeyFile:= 'C:\TwinCAT\3.1\Config\Certificates\AWS\private.pem.key';
    fbHttpClientAwsIot.nHostPort:= 8443;
END_IF

IF fbHttpClientAwsIot.bConfigured THEN
    fbHttpGetAwsIotShadow(bSend:=bGetAwsIotShadow, fbClient:=fbHttpClientAwsIot);
    fbHttpPostAwsIot(bSend:=bPostAwsIot, fbClient:=fbHttpClientAwsIot);
END_IF

fbHttpClientAwsIot.Execute();
```

6.1.3.1 Get

1. The function block is started by writing the variable bGetAwsIotShadow in the main program to TRUE.
2. The rising edge is then used to send a GET request with the IotHttpRequest function block.

3. After the request has been finished, the error handling is processed. When neither the function block itself nor the HTTP status code display an error, the JSON response from the webserver can be parsed within the PLC program.

```
FUNCTION_BLOCK FB_TestHTTP_Get_awsIotShadow
VAR_INPUT
    bSend               : BOOL;
END_VAR
VAR_IN_OUT
    fbClient            : FB_IotHttpClient;
END_VAR
VAR_OUTPUT
    bBusy               : BOOL;
    bError              : BOOL;
END_VAR
VAR
    fbRequest           : FB_IotHttpRequest;
    fbJson              : FB_JsonDomParser;
    nState              : UDINT;
    RisingEdge          : R_TRIG;

    bGetContentResult   : BOOL;
    sContent            : STRING(511);

    bGetJsonResult      : BOOL;
    jsonDoc             : SJsonValue;
    jsonVal             : SJsonValue;
    sResultValue        : STRING;

    nReqCount           : UDINT;
    nResCount           : UDINT;
    nValidResCount      : UDINT;
    nErrCount           : UDINT;
END_VAR
```

```
RisingEdge(CLK:= bSend);
CASE nState OF
0:
    IF RisingEdge.Q THEN
        IF fbRequest.SendRequest(sUri:= '/things/thingName/shadow', fbClient:= fbClient,
                                 eRequestType:= ETcIotHttpRequestType.HTTP_GET, 0, 0, 0) THEN
            nState:= 1;
            nReqCount:= nReqCount+1;
            bBusy:= TRUE;
            bError:= FALSE;
        END_IF
    END_IF
1:
    IF NOT fbRequest.bBusy THEN
        bError:= TRUE;
        IF NOT fbRequest.bError THEN
            bGetContentResult:= fbRequest.GetContent(pContent:= ADR(sContent),
                                                     nContentSize:= SIZEOF(sContent),
                                                     bSetNullTermination:= TRUE);
            IF fbRequest.nStatusCode >= 200 AND fbRequest.nStatusCode < 300 THEN
                bGetJsonResult:= FALSE;
                jsonDoc:= fbRequest.GetJsonDomContent(fbJson);
                IF jsonDoc <>0 THEN
                    ; // do something with the shadow document
                    nValidResCount:= nValidResCount+1;
                    bError:= FALSE;
                END_IF
                nResCount:= nResCount+1;
            END_IF
        END_IF
        nState:= 0;
        bBusy:= FALSE;
        IF bError THEN
            nErrCount:= nErrCount+1;
        END_IF
    END_IF
END_CASE
```

6.1.3.2 Post

1. The function block is started by writing the variable bPostAwsIot in the main program to TRUE.
2. The rising edge is then used to send a POST request with the IotHttpRequest function block.

3. After the request has been finished, the error handling is processed. When neither the function block itself nor the HTTP status code display an error, the JSON response from the webserver is parsed. The JSON response contains the message ‘OK’ if the POST request has been successful.

```
FUNCTION_BLOCK FB_TestHTTP_Post_awsIot
VAR_INPUT
    bSend               : BOOL;
END_VAR
VAR_IN_OUT
    fbClient            : FB_IotHttpClient;
END_VAR
VAR_OUTPUT
    bBusy               : BOOL;
    bError              : BOOL;
END_VAR
VAR
    fbRequest           : FB_IotHttpRequest;
    fbJson              : FB_JsonDomParser;
    fbPayload           : FB_JsonSaxWriter;
    nState              : UDINT;
    RisingEdge          : R_TRIG;

    bGetContentResult   : BOOL;
    sContent            : STRING(511);

    bGetJsonResult      : BOOL;
    jsonDoc             : SJsonValue;
    jsonVal             : SJsonValue;
    sResultValue        : STRING;

    nReqCount           : UDINT;
    nResCount           : UDINT;
    nValidResCount      : UDINT;
    nErrCount           : UDINT;
END_VAR
```

```
RisingEdge(CLK:= bSend);
CASE nState OF
0:
    IF RisingEdge.Q THEN
        fbPayload.StartObject();
        fbPayload.AddKey('message');
        fbPayload.AddReal(42.42);
        fbPayload.EndObject();
        sContent:= fbPayload.GetDocument();
        fbPayload.ResetDocument();
        IF fbRequest.SendRequest(sUri:= '/topics/mytopic?qos=1', fbClient:= fbClient,
                                 eRequestType:= ETcIotHttpRequestType.HTTP_POST,
                                 pContent:= ADR(sContent),
                                 nContentSize:= LEN2(ADR(sContent)), 0) THEN
            nState:= 1;
            nReqCount:= nReqCount+1;
            bBusy:= TRUE;
            bError:= FALSE;
        END_IF
    END_IF
1:
    IF NOT fbRequest.bBusy THEN
        bError:= TRUE;
        IF NOT fbRequest.bError THEN
            bGetContentResult:= fbRequest.GetContent(pContent:= ADR(sContent),
                                                     nContentSize:= SIZEOF(sContent),
                                                     bSetNullTermination:= TRUE);
            IF fbRequest.nStatusCode >= 200 AND fbRequest.nStatusCode < 300 THEN
                bGetJsonResult:= FALSE;
                jsonDoc:= fbRequest.GetJsonDomContent(fbJson);
                IF jsonDoc <>0 THEN
                    bGetJsonResult:= TRUE;
                    IF fbJson.HasMember(jsonDoc, 'message') THEN
                        jsonVal:= fbJson.FindMember(jsonDoc, 'message');
                        sResultValue:= fbJson.GetString(jsonVal);
                        IF sResultValue= 'OK' THEN
                            nValidResCount:= nValidResCount+1;
                            bError:= FALSE;
                        END_IF
                    END_IF
                END_IF
                nResCount:= nResCount+1;
            END_IF
        END_IF
        nState:= 0;
        bBusy:= FALSE;
        IF bError THEN
            nErrCount:= nErrCount+1;
        END_IF
    END_IF
END_CASE
```

6.1.4 Philips Hue

This part of the sample includes a PUT request to a Philips Hue Bridge (Required TwinCAT version: 3.1.4024.10). The REST API of the bridge enables the TwinCAT user to control Philips Hue devices out of the PLC. The user is required to have a Philips Hue Bridge available through his network and at least one device connected to it.

Basically, the HTTP client, in this case TwinCAT, sends different values as a JSON document to the bridge (e.g. saturation, brightness, state and color value for a bulb). The bridge answers with another JSON document which shows if the commands have been successful.

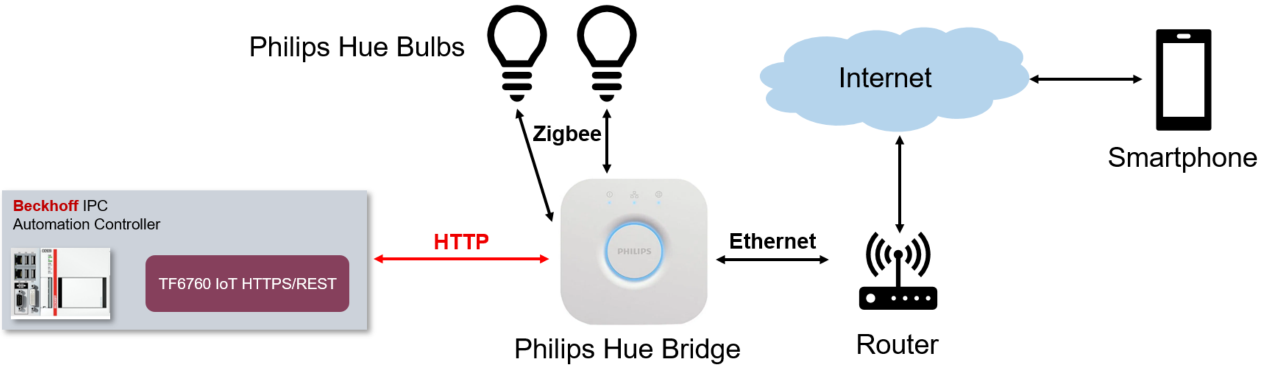


Fig. 3: Scheme PhilipsHue

Within the setup process, the Philips Hue Bridge randomly generates a username for a device that acts as HTTP client. This username can then be used for the communication with the bridge.

Philips delivers a getting started tutorial which shows how to find the bridge in a network and how to give a client application here: https://developers.meethue.com/develop/get-started-2/
The API description can be found on the same website, a registration is required for that.

The following sample implements on the one hand a single PUT request and on the other hand a toggeling PUT request that is used for a so-called “Blinking mode”. This mode triggers a Philips Hue Go lamp to change the color (between 0 and 65000) every time the timer output is set.

```
PROGRAM MAIN
VAR
    // trigger command execution for Philips Hue samples
    bPutPhilipsHue : BOOL;

    fbHttpClientPhilips Hue        : FB_IotHttpClient :=(sHostName:='172.17.x.x',
                                     bKeepAlive:=TRUE, tConnectionTimeout:=T#10S);

    fbHttpPutPhilipsHue            : FB_TestHTTP_Put_PhilipsHue;

    bBlinkingMode                  : BOOL;
    fbTimer                        : TON:=(PT:=T#500MS);
    nColor                         : UINT;
END_VAR
```

```
//init client parameters at startup
IF NOT fbHttpClientPhilipsHue.bConfigured THEN
    fbHttpClientPhilipsHue.nHostPort:= 80;
    fbHttpClientPhilipsHue.stTLS.bNoServerCertCheck:= FALSE;
END_IF

IF fbHttpClientPhilipsHue.bConfigured THEN
    fbHttpPutPhilipsHue(bSend:=bPutPhilipsHue, fbClient:=fbHttpClientPhilipsHue, nColor:=nColor);
END_IF
```

```
IF bBlinkingMode THEN
    fbTimer(IN:=NOT fbTimer.Q);
    IF fbTimer.Q THEN
        nColor:=nColor+5000;
        IF nColor=65000 THEN
            nColor:=0;
        END_IF
        IF bPutPhilipsHue THEN
            bPutPhilipsHue:=FALSE;
        ELSE
            bPutPhilipsHue:=TRUE;
        END_IF
    END_IF
END_IF

fbHttpClientPhilipsHue.Execute();
```

6.1.4.1 Put

1. The function block is started by writing the variable bPutPhilipsHue in the main program to TRUE.
2. The rising edge is then used to send a PUT request with the IotHttpRequest function block.
3. After the request has been finished, the error handling is processed. When neither the function block itself nor the HTTP status code display an error, the JSON response from the Philips Hue Bridge could be parsed. The JSON shows the user if the request has been successful.

```
FUNCTION_BLOCK FB_TestHTTP_Put_PhilipsHue
VAR_INPUT
    bSend                   : BOOL;
    nColor                  : UINT;
END_VAR
VAR_IN_OUT
    fbClient                : FB_IotHttpClient;
END_VAR
VAR_OUTPUT
    bBusy                   : BOOL;
    bError                  : BOOL;
END_VAR
VAR
    fbRequest               : FB_IotHttpRequest;
    fbJson                  : FB_JsonDomParser;
    fbJsonWriter            : FB_JsonSaxWriter;
    nState                  : UDINT;
    RisingEdge              : R_TRIG;

    bGetContentResult       : BOOL;
    sContent                : STRING(511);
    sSend                   : STRING(511);

    bGetJsonResult          : BOOL;
    jsonDoc                 : SJsonValue;
    jsonVal                 : SJsonValue;
    sResultValue            : STRING;

    nReqCount               : UDINT;
    nResCount               : UDINT;
    nValidResCount          : UDINT;
    nErrCount               : UDINT;

    fbTimer                 : TON;
    bLightOn                : BOOL;
END_VAR
```

```
RisingEdge(CLK:= bSend );

CASE nState OF
0:
        fbJsonWriter.StartObject();
        fbJsonWriter.AddKey('on');
        fbJsonWriter.AddBool(TRUE);
        fbJsonWriter.AddKey('sat');
        fbJsonWriter.AddUdint(50);
        fbJsonWriter.AddKey('bri');
        fbJsonWriter.AddUdint(100);
        fbJsonWriter.AddKey('hue');
        fbJsonWriter.AddUdint(nColor);
        fbJsonWriter.EndObject();
        sSend:=fbJsonWriter.GetDocument();
        fbJsonWriter.ResetDocument();
    IF RisingEdge.Q THEN
        IF fbRequest.SendRequest(sUri:='/api/qiyut7ubQQQv2xKJfFe9cVvIroUGHtIk2eYsIfGX/lights/1/
state', fbClient:=fbClient, eRequestType:=ETcIotHttpRequestType.HTTP_PUT,
            pContent:=ADR(sSend), nContentSize:=LEN2(ADR(sSend)), 0) THEN
            nState:= 1;
            nReqCount:= nReqCount+1;
            bBusy:= TRUE;
            bError:= FALSE;
        END_IF
    END_IF
1:
    IF NOT fbRequest.bBusy THEN
        bError:= TRUE;
        IF NOT fbRequest.bError THEN
            bGetContentResult:= fbRequest.GetContent(pContent:= ADR(sContent), nContentSize:= SIZEOF
(sContent), bSetNullTermination:= TRUE);
            IF fbRequest.nStatusCode >= 200 AND fbRequest.nStatusCode < 300 THEN
                bGetJsonResult:= FALSE;
                jsonDoc:= fbRequest.GetJsonDomContent(fbJson);
                IF jsonDoc <> 0 THEN
                    ; // do something with status response
                    nValidResCount:= nValidResCount+1;
                    bError:= FALSE;
                END_IF
                nResCount:= nResCount+1;
            END_IF
        END_IF
        nState:= 0;
        bBusy:= FALSE;
        IF bError THEN
            nErrCount:= nErrCount+1;
        END_IF
    END_IF
END_CASE
```

6.1.5 Telegram

This part of the sample includes a GET request to the REST API of the Telegram messenger (Required TwinCAT version: 3.1.4024.10). With the help of a so-called “Telegram Bot”, a TwinCAT user can send messages to a specific Telegram chat.

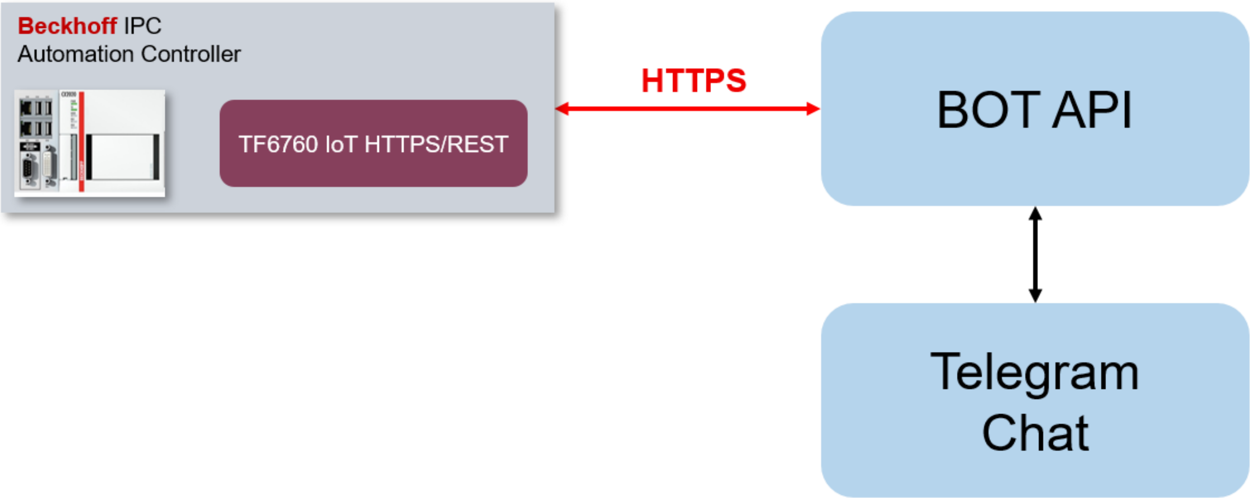
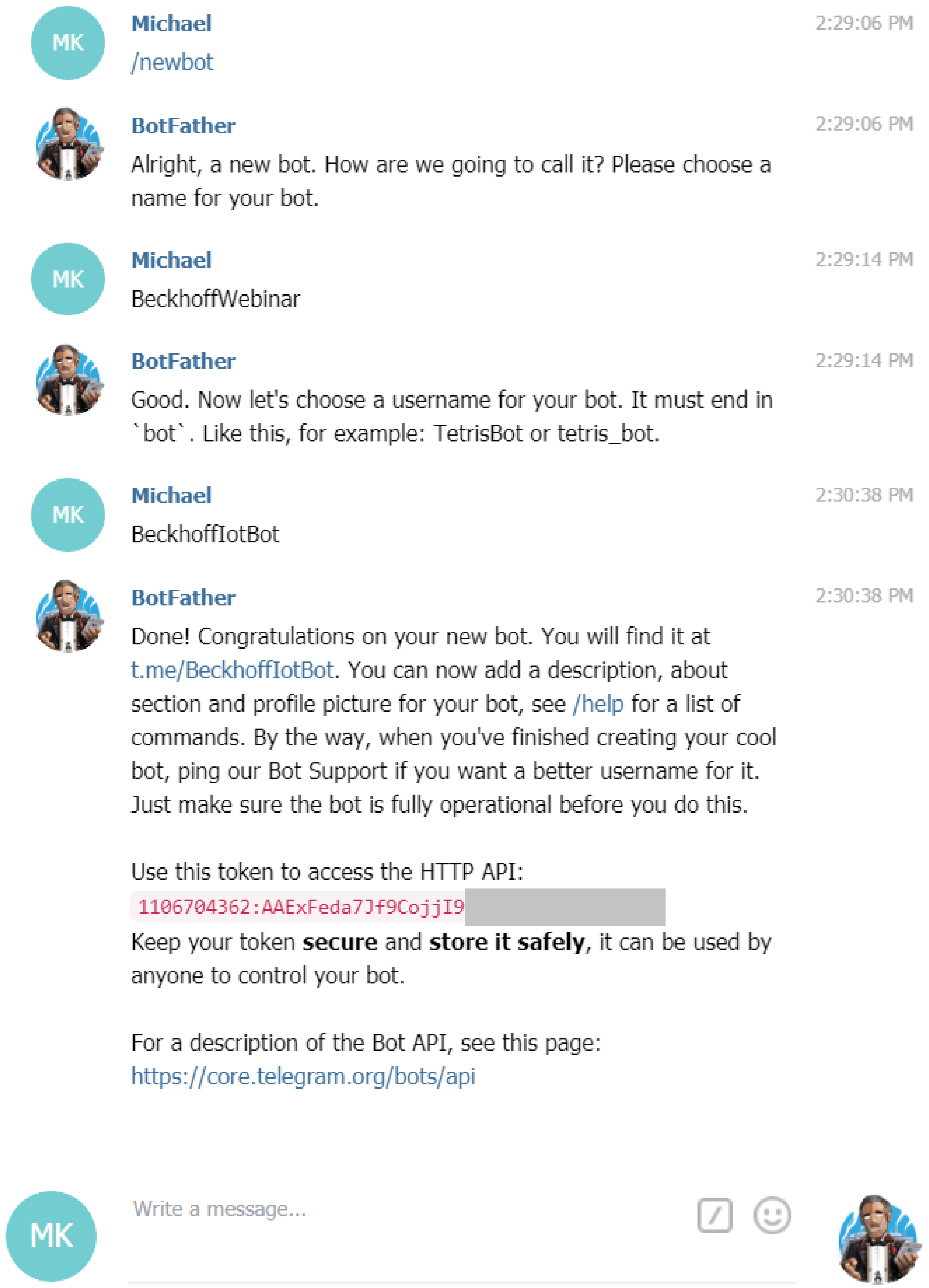


Fig. 4: Scheme Telegram

The first step of establishing a connection is the creation of the “Telegram-Bot”. Therefore, the user needs a Telegram account, from where he can contact the bot.



Michael 2:29:06 PM

/newbot

BotFather 2:29:06 PM

Alright, a new bot. How are we going to call it? Please choose a name for your bot.

Michael 2:29:14 PM

BeckhoffWebinar

BotFather 2:29:14 PM

Good. Now let's choose a username for your bot. It must end in `bot`. Like this, for example: TetrisBot or tetris_bot.

Michael 2:30:38 PM

BeckhoffIotBot

BotFather 2:30:38 PM

Done! Congratulations on your new bot. You will find it at t.me/BeckhoffIotBot. You can now add a description, about section and profile picture for your bot, see /help for a list of commands. By the way, when you've finished creating your cool bot, ping our Bot Support if you want a better username for it. Just make sure the bot is fully operational before you do this.

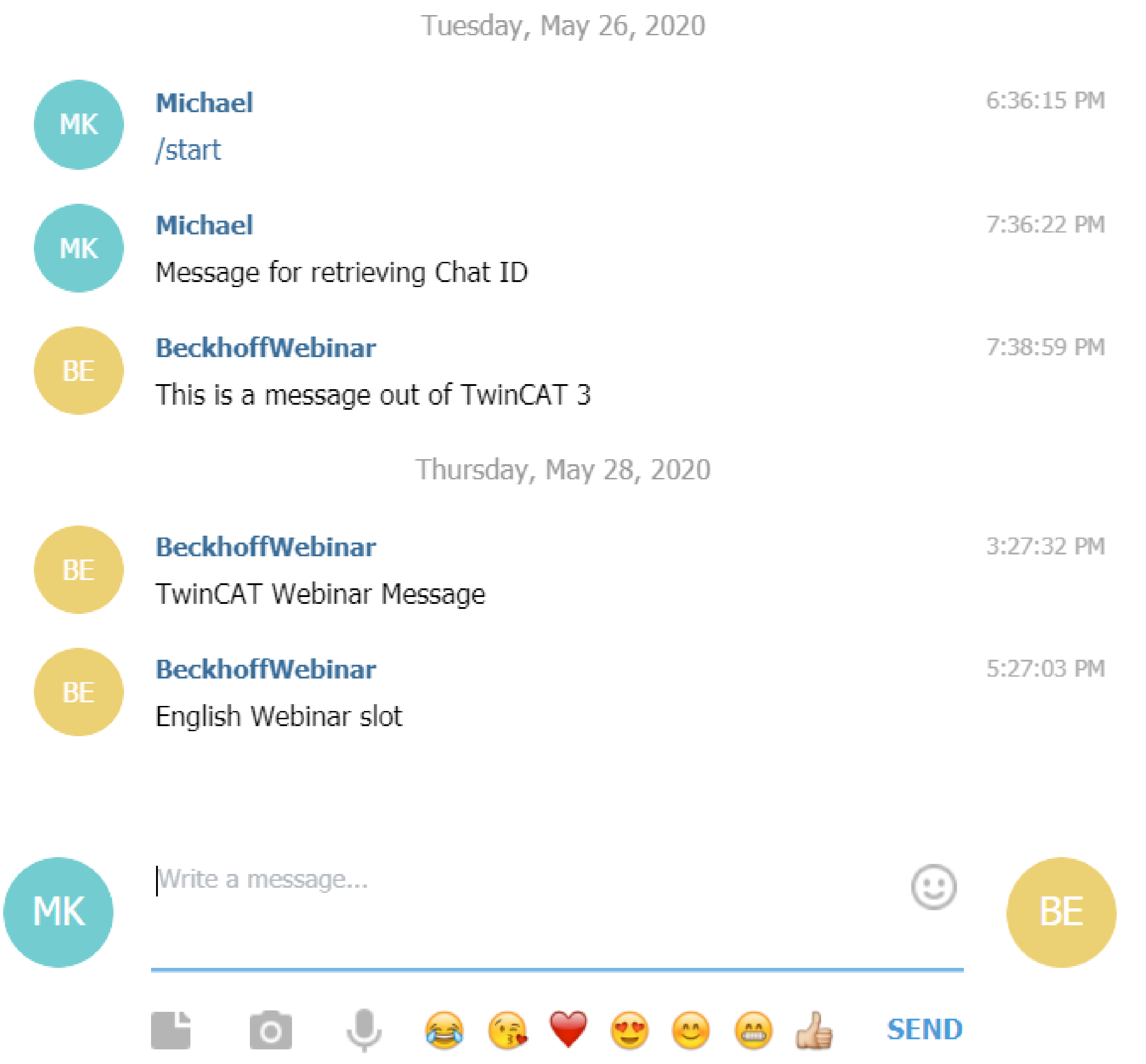
Use this token to access the HTTP API:
`1106704362:AAExFeda7Jf9CojjI9`[redacted]
Keep your token **secure** and **store it safely**, it can be used by anyone to control your bot.

For a description of the Bot API, see this page:
https://core.telegram.org/bots/api

Fig. 5: Telegram BotFather

The bot will return an API token, which has to be used when sending requests out of TwinCAT to the Telegram API. With access to this token, a HTTP client can control every function that the bot delivers. A description of the BOT API can be found here: https://core.telegram.org/bots/api

When sending messages from a TwinCAT PLC to a Telegram Bot, it can look like in the following picture. In this sample a message from the Telegram browser version to the bot is used to request the Chat ID.



Tuesday, May 26, 2020

MK Michael 6:36:15 PM
/start

MK Michael 7:36:22 PM
Message for retrieving Chat ID

BE BeckhoffWebinar 7:38:59 PM
This is a message out of TwinCAT 3

Thursday, May 28, 2020

BE BeckhoffWebinar 3:27:32 PM
TwinCAT Webinar Message

BE BeckhoffWebinar 5:27:03 PM
English Webinar slot

Fig. 6: Telegram Chat

```
PROGRAM MAIN
VAR
    // trigger command execution for Telegram samples
    bGetTelegram                       : BOOL;

    fbHttpClientTelegram               : FB_IotHttpClient :=(sHostName:='api.telegram.org',
                                       bKeepAlive:=FALSE, tConnectionTimeout:=T#10S);

    fbHttpGetTelegram                  : FB_TestHTTP_Get_Telegram;
    sMessage                           : STRING(500);
END_VAR
```

```
//init client parameters at startup
IF NOT fbHttpClientTelegram.bConfigured THEN
    fbHttpClientTelegram.stTLS.sCA:='C:\TwinCAT\3.1\Config\Certificates\TelegramRoot.cer';
    fbHttpClientTelegram.nHostPort:=443;
    fbHttpClientTelegram.stTLS.bNoServerCertCheck:=FALSE;
END_IF
```

```
IF fbHttpClientTelegram.bConfigured THEN
    fbHttpGetTelegram(bSend:=bGetTelegram, fbClient:=fbHttpClientTelegram, sMessage:=sMessage);
END_IF

fbHttpClientTelegram.Execute();
```

6.1.5.1 Get

1. The function block is started by setting the variable bGetTelegram in the main program to TRUE.
2. The rising edge is then used to send a GET request with the function block IotHttpRequest.
3. Once the request is complete, an error handling routine is invoked. If both the function block itself and the HTTP status code are error-free, the JSON response can be parsed by the web server in the PLC program.

```
FUNCTION_BLOCK FB_TestHTTP_Get_Telegram
VAR_INPUT
    bSend               : BOOL;
    sMessage            : STRING(500);
END_VAR
VAR_IN_OUT
    fbClient            : FB_IotHttpClient;
END_VAR
VAR_OUTPUT
    bBusy               : BOOL;
    bError              : BOOL;
END_VAR
VAR
    fbRequest           : FB_IotHttpRequest;
    fbJson              : FB_JsonDomParser;
    nState              : UDINT;
    RisingEdge          : R_TRIG;

    bGetContentResult   : BOOL;
    sContent            : STRING(511);

    bGetJsonResult      : BOOL;
    jsonDoc             : SJsonValue;
    jsonVal             : SJsonValue;
    sResultValue        : STRING;

    nReqCount           : UDINT;
    nResCount           : UDINT;
    nValidResCount      : UDINT;
    nErrCount           : UDINT;
    fbFormatString      : FB_FormatString;
    sBotApiKey          : STRING(500):='1106704362:AAExFeda7Jf9CojjI9whLEzPeE4KDlDzEnf';
    sChatId             : STRING(500):='1140427258';
    sConMessage         : STRING(500);
END_VAR
RisingEdge(CLK:= bSend );
```

```
fbFormatString(
    sFormat := '/bot%s/sendMessage?chat_id=%s&text=%s',
    arg1    := F_STRING(sBotApiKey),
    arg2    := F_STRING(sChatId),
    arg3    := F_STRING(sMessage),
    bError  => ,
    nErrId  => ,
    sOut    => sConMessage);

CASE nState OF
0:
    IF RisingEdge.Q THEN
        IF fbRequest.SendRequest(sUri:=sConMessage, fbClient:=fbClient, eRequestType:=ETcIotHttpRequ
estType.HTTP_Get, 0, 0, 0) THEN
            nState:= 1;
            nReqCount:= nReqCount+1;
            bBusy:= TRUE;
            bError:= FALSE;
        END_IF
    END_IF
1:
    IF NOT fbRequest.bBusy THEN
        bError:= TRUE;
        IF NOT fbRequest.bError THEN
```

```
            bGetContentResult:= fbRequest.GetContent(pContent:= ADR(sContent), nContentSize:= SIZEOF
(sContent), bSetNullTermination:= TRUE);
            IF fbRequest.nStatusCode >= 200 AND fbRequest.nStatusCode < 300 THEN
                bGetJsonResult:= FALSE;
                jsonDoc:= fbRequest.GetJsonDomContent(fbJson);
                IF jsonDoc <> 0 THEN
                    ; // do something with the response
                    nValidResCount:= nValidResCount+1;
                    bError:= FALSE;
                END_IF
                nResCount:= nResCount+1;
            END_IF
        END_IF
        nState:= 0;
        bBusy:= FALSE;
        IF bError THEN
            nErrCount:= nErrCount+1;
        END_IF
    END_IF
END_CASE
```

6.1.6 AWS Service Configuration

This sample shows how to access services of the cloud provider Amazon Web Services (AWS) from the PLC using a GET request (required TwinCAT version: 3.1.4024.12). The REST API provides all the functionality that a user would have in the AWS management console. For example, virtual machines can be instantiated or configured using the AWS EC2 service.

The chapter AWS Signature Version 4 [▶ 19] briefly touches on the background of the AWS signing function. For more information please refer to the AWS documentation.

As already described in the chapter URL redirects [▶ 20], the IoT driver does not evaluate URL redirects. Therefore, when accessing the local endpoint of an AWS data center, the exact address must always be specified. Again referring to the EC2 services sample, this means that a user cannot connect from TwinCAT to *ec2.amazonaws.com* but must directly provide the region in the link: *ec2.eu-central-1.amazonaws.com*.

A possible approach would be: Fetch all available regions via a location-independent AWS REST API function and extract the region endpoint for the desired region from it. This would have the advantage over static programming of the endpoint that changes in the endpoint URL of a region would not result in changes in the program code.

```
PROGRAM MAIN
VAR
    // trigger command execution for AWS Sig V4 samples
    bGetAWSSigV4                    : BOOL;

    fbHttpClientAWSSigV4            : FB_IotHttpClient :=(sHostName:='ec2.us-east-1.amazonaws.com',
                                    bKeepAlive:=FALSE, tConnectionTimeout:=T#10S);

    fbHttpGetAWSSigV4               : FB_TestHTTP_Get_AwsSigV4;
END_VAR
```

```
//init client parameters at startup
IF NOT fbHttpClientAWSSigV4.bConfigured THEN
    fbHttpClientAWSSigV4.nHostPort:=443;
    fbHttpClientAWSSigV4.stTLS.bNoServerCertCheck:=TRUE;
END_IF

IF fbHttpClientAWSSigV4.bConfigured THEN
    fbHttpGetAWSSigV4(bSend:= bGetAWSSigV4, fbClient:= fbHttpClientAWSSigV4);
END_IF

fbHttpClientAWSSigV4.Execute();
```

6.1.6.1 Get

1. The function block is started by setting the variable bGetAWSSigV4 in the main program to TRUE.
2. The rising edge is then used to send a GET request with the function block IotHttpRequest.

3. Once the request is complete, an error handling routine is invoked. If both the function block itself and the HTTP status code are error-free, the XML response can be parsed by the web server in the PLC program (the response of the AWS REST API can become quite long, so sufficient memory should be reserved for this purpose, and it may be advisable to test the procedure with an HTTP test client).

Alphabetical sorting required

The request URL must be sorted alphabetically, ditto the signed header. In the case of the signed header, this is handled by the IoT driver; in the case of the request URL, it is the responsibility of the user, otherwise an error message will be issued by the HTTP server.

```
FUNCTION_BLOCK FB_TestHTTP_Get_AWSSigV4
VAR_INPUT
    bSend                : BOOL;
END_VAR
VAR_IN_OUT
    fbClient             : FB_IotHttpClient;
END_VAR
VAR_OUTPUT
    bBusy                : BOOL;
    bError               : BOOL;
END_VAR
VAR
    fbRequest            : FB_IotHttpRequest;
    fbSig4Header         : FB_IotHttpAwsSigV4HeaderFieldMap;
    nState               : UDINT;
    RisingEdge           : R_TRIG;

    bGetContentResult    : BOOL;
    sContent             : STRING(5000);

    bGetJsonResult       : BOOL;
    sResultValue         : STRING;

    nReqCount            : UDINT;
    nResCount            : UDINT;
    nValidResCount       : UDINT;
    nErrCount            : UDINT;
    sRequestUrl          : STRING(500):='?
Action=RunInstances&ImageId=ami-0229f7666f517b31e&InstanceType=t2.small&KeyName=TestKDB&MaxCount=1&M
inCount=1&Version=2016-11-15';
    //sRequestUrl        : STRING(500):='?Action=DescribeInstances&Version=2016-11-15';
    sService             : STRING:='ec2';
    sRegion              : STRING:='us-east-1';
    sAccessKey           : STRING:='censored';
    sSecretKey           : STRING:='censored';
    sSignedHeaders       : STRING:='host;x-amz-date';
    bSetParameter        : BOOL;
END_VAR
```

```
RisingEdge(CLK:= bSend );

CASE nState OF
0:
    IF RisingEdge.Q THEN
    bSetParameter:=fbSig4Header.SetParameter(sService, sRegion, sAccessKey, sSecretKey, sSignedHeade
rs);
        IF fbRequest.SendRequest(sUri:=sConMessage, fbClient:=fbClient, eRequestType:=ETcIotHttpRequ
estType.HTTP_Get, 0, 0, fbHeader:=fbSig4Header) THEN
            nState:= 1;
            nReqCount:= nReqCount+1;
            bBusy:= TRUE;
            bError:= FALSE;
        END_IF
        bSetParameter:=FALSE;
    END_IF
1:
    IF NOT fbRequest.bBusy THEN
        bError:= TRUE;
        IF NOT fbRequest.bError THEN
            bGetContentResult:= fbRequest.GetContent(pContent:= ADR(sContent), nContentSize:= SIZEOF
(sContent), bSetNullTermination:= TRUE);
            IF fbRequest.nStatusCode >= 200 AND fbRequest.nStatusCode < 300 THEN
                nResCount:= nResCount+1;
                // do something with the XML response
                bError:=FALSE;
            END_IF
        END_IF
        nState:= 0;
        bBusy:= FALSE;
        IF bError THEN
            nErrCount:= nErrCount+1;
        END_IF
    END_IF
END_CASE
```

7 Appendix

7.1 ADS Return Codes

Grouping of error codes:
Global error codes: ADS Return Codes [▶ 143]... (0x9811_0000 ...)
Router error codes: ADS Return Codes [▶ 143]... (0x9811_0500 ...)
General ADS errors: ADS Return Codes [▶ 144]... (0x9811_0700 ...)
RTime error codes: ADS Return Codes [▶ 146]... (0x9811_1000 ...)

Global error codes

Hex	Dec	HRESULT	Name	Description
0x0	0	0x98110000	ERR_NOERROR	No error.
0x1	1	0x98110001	ERR_INTERNAL	Internal error.
0x2	2	0x98110002	ERR_NORTIME	No real time.
0x3	3	0x98110003	ERR_ALLOCLOCKEDMEM	Allocation locked – memory error.
0x4	4	0x98110004	ERR_INSERTMAILBOX	Mailbox full – the ADS message could not be sent. Reducing the number of ADS messages per cycle will help.
0x5	5	0x98110005	ERR_WRONGRECEIVEHMSG	Wrong HMSG.
0x6	6	0x98110006	ERR_TARGETPORTNOTFOUND	Target port not found – ADS server is not started or is not reachable.
0x7	7	0x98110007	ERR_TARGETMACHINENOTFOUND	Target computer not found – AMS route was not found.
0x8	8	0x98110008	ERR_UNKNOWNCMDID	Unknown command ID.
0x9	9	0x98110009	ERR_BADTASKID	Invalid task ID.
0xA	10	0x9811000A	ERR_NOIO	No IO.
0xB	11	0x9811000B	ERR_UNKNOWNAMSCMD	Unknown AMS command.
0xC	12	0x9811000C	ERR_WIN32ERROR	Win32 error.
0xD	13	0x9811000D	ERR_PORTNOTCONNECTED	Port not connected.
0xE	14	0x9811000E	ERR_INVALIDAMSLENGTH	Invalid AMS length.
0xF	15	0x9811000F	ERR_INVALIDAMSNETID	Invalid AMS Net ID.
0x10	16	0x98110010	ERR_LOWINSTLEVEL	Installation level is too low –TwinCAT 2 license error.
0x11	17	0x98110011	ERR_NODEBUGINTAVAILABLE	No debugging available.
0x12	18	0x98110012	ERR_PORTDISABLED	Port disabled – TwinCAT system service not started.
0x13	19	0x98110013	ERR_PORTALREADYCONNECTED	Port already connected.
0x14	20	0x98110014	ERR_AMSSYNC_W32ERROR	AMS Sync Win32 error.
0x15	21	0x98110015	ERR_AMSSYNC_TIMEOUT	AMS Sync Timeout.
0x16	22	0x98110016	ERR_AMSSYNC_AMSERROR	AMS Sync error.
0x17	23	0x98110017	ERR_AMSSYNC_NOINDEXINMAP	No index map for AMS Sync available.
0x18	24	0x98110018	ERR_INVALIDAMSPORT	Invalid AMS port.
0x19	25	0x98110019	ERR_NOMEMORY	No memory.
0x1A	26	0x9811001A	ERR_TCPSEND	TCP send error.
0x1B	27	0x9811001B	ERR_HOSTUNREACHABLE	Host unreachable.
0x1C	28	0x9811001C	ERR_INVALIDAMSFRAGMENT	Invalid AMS fragment.
0x1D	29	0x9811001D	ERR_TLSSEND	TLS send error – secure ADS connection failed.
0x1E	30	0x9811001E	ERR_ACCESSDENIED	Access denied – secure ADS access denied.

Router error codes

Hex	Dec	HRESULT	Name	Description
0x500	1280	0x98110500	ROUTERERR_NOLOCKEDMEMORY	Locked memory cannot be allocated.
0x501	1281	0x98110501	ROUTERERR_RESIZEMEMORY	The router memory size could not be changed.
0x502	1282	0x98110502	ROUTERERR_MAILBOXFULL	The mailbox has reached the maximum number of possible messages.
0x503	1283	0x98110503	ROUTERERR_DEBUGBOXFULL	The Debug mailbox has reached the maximum number of possible messages.
0x504	1284	0x98110504	ROUTERERR_UNKNOWNPORTTYPE	The port type is unknown.
0x505	1285	0x98110505	ROUTERERR_NOTINITIALIZED	The router is not initialized.
0x506	1286	0x98110506	ROUTERERR_PORTALREADYINUSE	The port number is already assigned.
0x507	1287	0x98110507	ROUTERERR_NOTREGISTERED	The port is not registered.
0x508	1288	0x98110508	ROUTERERR_NOMOREQUEUES	The maximum number of ports has been reached.
0x509	1289	0x98110509	ROUTERERR_INVALIDPORT	The port is invalid.
0x50A	1290	0x9811050A	ROUTERERR_NOTACTIVATED	The router is not active.
0x50B	1291	0x9811050B	ROUTERERR_FRAGMENTBOXFULL	The mailbox has reached the maximum number for fragmented messages.
0x50C	1292	0x9811050C	ROUTERERR_FRAGMENTTIMEOUT	A fragment timeout has occurred.
0x50D	1293	0x9811050D	ROUTERERR_TOBEREMOVED	The port is removed.

General ADS error codes

Hex	Dec	HRESULT	Name	Description
0x700	1792	0x98110700	ADSERR_DEVICE_ERROR	General device error.
0x701	1793	0x98110701	ADSERR_DEVICE_SRVNOTSUPP	Service is not supported by the server.
0x702	1794	0x98110702	ADSERR_DEVICE_INVALIDGRP	Invalid index group.
0x703	1795	0x98110703	ADSERR_DEVICE_INVALIDOFFSET	Invalid index offset.
0x704	1796	0x98110704	ADSERR_DEVICE_INVALIDACCESS	Reading or writing not permitted.
0x705	1797	0x98110705	ADSERR_DEVICE_INVALIDSIZE	Parameter size not correct.
0x706	1798	0x98110706	ADSERR_DEVICE_INVALIDDATA	Invalid data values.
0x707	1799	0x98110707	ADSERR_DEVICE_NOTREADY	Device is not ready to operate.
0x708	1800	0x98110708	ADSERR_DEVICE_BUSY	Device is busy.
0x709	1801	0x98110709	ADSERR_DEVICE_INVALIDCONTEXT	Invalid operating system context. This can result from use of ADS blocks in different tasks. It may be possible to resolve this through multitasking synchronization in the PLC.
0x70A	1802	0x9811070A	ADSERR_DEVICE_NOMEMORY	Insufficient memory.
0x70B	1803	0x9811070B	ADSERR_DEVICE_INVALIDPARM	Invalid parameter values.
0x70C	1804	0x9811070C	ADSERR_DEVICE_NOTFOUND	Not found (files, ...).
0x70D	1805	0x9811070D	ADSERR_DEVICE_SYNTAX	Syntax error in file or command.
0x70E	1806	0x9811070E	ADSERR_DEVICE_INCOMPATIBLE	Objects do not match.
0x70F	1807	0x9811070F	ADSERR_DEVICE_EXISTS	Object already exists.
0x710	1808	0x98110710	ADSERR_DEVICE_SYMBOLNOTFOUND	Symbol not found.
0x711	1809	0x98110711	ADSERR_DEVICE_SYMBOLVERSIONINVALID	Invalid symbol version. This can occur due to an online change. Create a new handle.
0x712	1810	0x98110712	ADSERR_DEVICE_INVALIDSTATE	Device (server) is in invalid state.
0x713	1811	0x98110713	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransMode not supported.
0x714	1812	0x98110714	ADSERR_DEVICE_NOTIFYHNDINVALID	Notification handle is invalid.
0x715	1813	0x98110715	ADSERR_DEVICE_CLIENTUNKNOWN	Notification client not registered.
0x716	1814	0x98110716	ADSERR_DEVICE_NOMOREHDLS	No further handle available.
0x717	1815	0x98110717	ADSERR_DEVICE_INVALIDWATCHSIZE	Notification size too large.
0x718	1816	0x98110718	ADSERR_DEVICE_NOTINIT	Device not initialized.
0x719	1817	0x98110719	ADSERR_DEVICE_TIMEOUT	Device has a timeout.
0x71A	1818	0x9811071A	ADSERR_DEVICE_NOINTERFACE	Interface query failed.
0x71B	1819	0x9811071B	ADSERR_DEVICE_INVALIDINTERFACE	Wrong interface requested.
0x71C	1820	0x9811071C	ADSERR_DEVICE_INVALIDCLSID	Class ID is invalid.
0x71D	1821	0x9811071D	ADSERR_DEVICE_INVALIDOBJID	Object ID is invalid.
0x71E	1822	0x9811071E	ADSERR_DEVICE_PENDING	Request pending.
0x71F	1823	0x9811071F	ADSERR_DEVICE_ABORTED	Request is aborted.
0x720	1824	0x98110720	ADSERR_DEVICE_WARNING	Signal warning.
0x721	1825	0x98110721	ADSERR_DEVICE_INVALIDARRAYIDX	Invalid array index.
0x722	1826	0x98110722	ADSERR_DEVICE_SYMBOLNOTACTIVE	Symbol not active.
0x723	1827	0x98110723	ADSERR_DEVICE_ACCESSDENIED	Access denied.
0x724	1828	0x98110724	ADSERR_DEVICE_LICENSENOTFOUND	Missing license.
0x725	1829	0x98110725	ADSERR_DEVICE_LICENSEEXPIRED	License expired.
0x726	1830	0x98110726	ADSERR_DEVICE_LICENSEEXCEEDED	License exceeded.
0x727	1831	0x98110727	ADSERR_DEVICE_LICENSEINVALID	Invalid license.
0x728	1832	0x98110728	ADSERR_DEVICE_LICENSESYSTEMID	License problem: System ID is invalid.
0x729	1833	0x98110729	ADSERR_DEVICE_LICENSENOTIMELIMIT	License not limited in time.
0x72A	1834	0x9811072A	ADSERR_DEVICE_LICENSEFUTUREISSUE	Licensing problem: time in the future.
0x72B	1835	0x9811072B	ADSERR_DEVICE_LICENSETIMETOLONG	License period too long.
0x72C	1836	0x9811072C	ADSERR_DEVICE_EXCEPTION	Exception at system startup.
0x72D	1837	0x9811072D	ADSERR_DEVICE_LICENSEDUPLICATED	License file read twice.
0x72E	1838	0x9811072E	ADSERR_DEVICE_SIGNATUREINVALID	Invalid signature.
0x72F	1839	0x9811072F	ADSERR_DEVICE_CERTIFICATEINVALID	Invalid certificate.
0x730	1840	0x98110730	ADSERR_DEVICE_LICENSEOEMNOTFOUND	Public key not known from OEM.
0x731	1841	0x98110731	ADSERR_DEVICE_LICENSERESTRICTED	License not valid for this system ID.
0x732	1842	0x98110732	ADSERR_DEVICE_LICENSEDEMODENIED	Demo license prohibited.
0x733	1843	0x98110733	ADSERR_DEVICE_INVALIDFNCID	Invalid function ID.
0x734	1844	0x98110734	ADSERR_DEVICE_OUTOFRANGE	Outside the valid range.
0x735	1845	0x98110735	ADSERR_DEVICE_INVALIDALIGNMENT	Invalid alignment.
0x736	1846	0x98110736	ADSERR_DEVICE_LICENSEPLATFORM	Invalid platform level.

Hex	Dec	HRESULT	Name	Description
0x737	1847	0x98110737	ADSERR_DEVICE_FORWARD_PL	Context – forward to passive level.
0x738	1848	0x98110738	ADSERR_DEVICE_FORWARD_DL	Context – forward to dispatch level.
0x739	1849	0x98110739	ADSERR_DEVICE_FORWARD_RT	Context – forward to real time.
0x740	1856	0x98110740	ADSERR_CLIENT_ERROR	Client error.
0x741	1857	0x98110741	ADSERR_CLIENT_INVALIDPARM	Service contains an invalid parameter.
0x742	1858	0x98110742	ADSERR_CLIENT_LISTEMPTY	Polling list is empty.
0x743	1859	0x98110743	ADSERR_CLIENT_VARUSED	Var connection already in use.
0x744	1860	0x98110744	ADSERR_CLIENT_DUPLINVOKEID	The called ID is already in use.
0x745	1861	0x98110745	ADSERR_CLIENT_SYNCTIMEOUT	Timeout has occurred – the remote terminal is not responding in the specified ADS timeout. The route setting of the remote terminal may be configured incorrectly.
0x746	1862	0x98110746	ADSERR_CLIENT_W32ERROR	Error in Win32 subsystem.
0x747	1863	0x98110747	ADSERR_CLIENT_TIMEOUTINVALID	Invalid client timeout value.
0x748	1864	0x98110748	ADSERR_CLIENT_PORTNOTOPEN	Port not open.
0x749	1865	0x98110749	ADSERR_CLIENT_NOAMSADDR	No AMS address.
0x750	1872	0x98110750	ADSERR_CLIENT_SYNCINTERNAL	Internal error in Ads sync.
0x751	1873	0x98110751	ADSERR_CLIENT_ADDHASH	Hash table overflow.
0x752	1874	0x98110752	ADSERR_CLIENT_REMOVEHASH	Key not found in the table.
0x753	1875	0x98110753	ADSERR_CLIENT_NOMORESYM	No symbols in the cache.
0x754	1876	0x98110754	ADSERR_CLIENT_SYNCRESINVALID	Invalid response received.
0x755	1877	0x98110755	ADSERR_CLIENT_SYNCPORTLOCKED	Sync Port is locked.

RTime error codes

Hex	Dec	HRESULT	Name	Description
0x1000	4096	0x98111000	RTERR_INTERNAL	Internal error in the real-time system.
0x1001	4097	0x98111001	RTERR_BADTIMERPERIODS	Timer value is not valid.
0x1002	4098	0x98111002	RTERR_INVALIDTASKPTR	Task pointer has the invalid value 0 (zero).
0x1003	4099	0x98111003	RTERR_INVALIDSTACKPTR	Stack pointer has the invalid value 0 (zero).
0x1004	4100	0x98111004	RTERR_PRIOEXISTS	The request task priority is already assigned.
0x1005	4101	0x98111005	RTERR_NOMORETCB	No free TCB (Task Control Block) available. The maximum number of TCBs is 64.
0x1006	4102	0x98111006	RTERR_NOMORESEMAS	No free semaphores available. The maximum number of semaphores is 64.
0x1007	4103	0x98111007	RTERR_NOMOREQUEUES	No free space available in the queue. The maximum number of positions in the queue is 64.
0x100D	4109	0x9811100D	RTERR_EXTIRQALREADYDEF	An external synchronization interrupt is already applied.
0x100E	4110	0x9811100E	RTERR_EXTIRQNOTDEF	No external sync interrupt applied.
0x100F	4111	0x9811100F	RTERR_EXTIRQINSTALLFAILED	Application of the external synchronization interrupt has failed.
0x1010	4112	0x98111010	RTERR_IRQLNOTLESSOREQUAL	Call of a service function in the wrong context
0x1017	4119	0x98111017	RTERR_VMXNOTSUPPORTED	Intel VT-x extension is not supported.
0x1018	4120	0x98111018	RTERR_VMXDISABLED	Intel VT-x extension is not enabled in the BIOS.
0x1019	4121	0x98111019	RTERR_VMXCONTROLSMISSING	Missing function in Intel VT-x extension.
0x101A	4122	0x9811101A	RTERR_VMXENABLEFAILS	Activation of Intel VT-x fails.

Specific positive HRESULT Return Codes:

HRESULT	Name	Description
0x0000_0000	S_OK	No error.
0x0000_0001	S_FALSE	No error. Example: successful processing, but with a negative or incomplete result.
0x0000_0203	S_PENDING	No error. Example: successful processing, but no result is available yet.
0x0000_0256	S_WATCHDOG_TIMEOUT	No error. Example: successful processing, but a timeout occurred.

TCP Winsock error codes

Hex	Dec	Name	Description
0x274C	10060	WSAETIMEDOUT	A connection timeout has occurred - error while establishing the connection, because the remote terminal did not respond properly after a certain period of time, or the established connection could not be maintained because the connected host did not respond.
0x274D	10061	WSAECONNREFUSED	Connection refused - no connection could be established because the target computer has explicitly rejected it. This error usually results from an attempt to connect to a service that is inactive on the external host, that is, a service for which no server application is running.
0x2751	10065	WSAEHOSTUNREACH	No route to host - a socket operation referred to an unavailable host.
More Winsock error codes: Win32 error codes			

7.2 Enumerations

7.2.1 ETcIotHttpRequestError

```
TYPE ETcIotHttpRequestError:
(
    HTTP_REQ_ERR_BUSY:=-1,
    HTTP_REQ_ERR_SUCCESS:=0,
    HTTP_REQ_ERR_NOMEM:=1,
    HTTP_REQ_ERR_CREATE_PROTOCOL:=2,
    HTTP_REQ_ERR_CONN_INVAL:=3,
    HTTP_REQ_ERR_NO_CONN:=4,
    HTTP_REQ_ERR_CONN_REFUSED:=5,
    HTTP_REQ_ERR_NOT_FOUND:=6,
    HTTP_REQ_ERR_CONN_LOST:=7,
    HTTP_REQ_ERR_TLS:=8,
    HTTP_REQ_ERR_NOT_SUPPORTED:=10,
    HTTP_REQ_ERR_AUTH:=11,
    HTTP_REQ_ERR_ACL_DENIED:=12,
    HTTP_REQ_ERR_UNKNOWN:=13,
    HTTP_REQ_ERR_ERRNO:=14,
    HTTP_REQ_ERR_EAI:=15,
    HTTP_REQ_ERR_PROXY:=16,
    HTTP_REQ_ERR_TLS_CA_NOTFOUND:=17,
    HTTP_REQ_ERR_TLS_CERT_NOTFOUND:=18,
    HTTP_REQ_ERR_TLS_KEY_NOTFOUND:=19,
    HTTP_REQ_ERR_TLS_CA_INVALID:=20,
    HTTP_REQ_ERR_TLS_CERT_INVALID:=21,
    HTTP_REQ_ERR_TLS_KEY_INVALID:=22,
    HTTP_REQ_ERR_TLS_VERIFY_FAIL:=23,
    HTTP_REQ_ERR_TLS_SETUP:=24,
    HTTP_REQ_ERR_TLS_HANDSHAKE_FAIL:=25,
    HTTP_REQ_ERR_TLS_CIPHER_INVALID:=26,
    HTTP_REQ_ERR_TLS_VERSION_INVALID:=27,
    HTTP_REQ_ERR_TLS_PSK_INVALID:=28,
    HTTP_REQ_ERR_TLS_CRL_NOTFOUND:=29,
    HTTP_REQ_ERR_TLS_CRL_INVALID:=30,
    HTTP_REQ_ERR_FINALIZE_DISCONNECT:=31,
    HTTP_REQ_ERR_BIND:=32,
    HTTP_REQ_ERR_BIND_ADDR_INUSE:=33,
    HTTP_REQ_ERR_BIND_ADDR_INVAL:=34,
    HTTP_REQ_ERR_CREATE:=35,
    HTTP_REQ_ERR_CREATE_TYPE:=36,
    HTTP_REQ_ERR_CONN:=37,
    HTTP_REQ_ERR_CONN_TIMEDOUT:=38,
    HTTP_REQ_ERR_CONN_HOSTUNREACH:=39,
    HTTP_REQ_ERR_TLS_CERT_EXPIRED:=40,
    HTTP_REQ_ERR_TLS_CN_MISMATCH:=41,
    HTTP_REQ_ERR_INV_PARAM:=1000,
    HTTP_REQ_ERR_FIFO_FULL:=1001,
    HTTP_REQ_ERR_TCP_SEND:=1002,
    HTTP_REQ_ERR_CANCELLED:=1003,
    HTTP_REQ_ERR_RESPONSE_TIMEDOUT:=1004,
    HTTP_REQ_ERR_INV_HDR_SIZE:=1005,
    HTTP_REQ_ERR_INV_ENCODING:=1006,
    HTTP_REQ_ERR_INV_CONTENT_SIZE:=1007,
    HTTP_REQ_ERR_INV_CHUNK_SIZE:=1008,
    HTTP_REQ_ERR_PARSE_HDR:=1100,
    HTTP_REQ_ERR_PARSE_HDR_FIELD:=1101,
    HTTP_REQ_ERR_PARSE_HDR_FIELD_NAME:=1102,
    HTTP_REQ_ERR_PARSE_HDR_FIELD_VAL:=1103,
    HTTP_REQ_ERR_PARSE_STATUS_LINE:=1104,
    HTTP_REQ_ERR_PARSE_CHUNK:=1105,
    HTTP_REQ_ERR_PARSE_CHUNK_SIZE:=1106,
    HTTP_REQ_ERR_PARSE_CHUNK_EXT_NAME:=1107,
    HTTP_REQ_ERR_PARSE_CHUNK_EXT_VAL:=1108,
    HTTP_REQ_ERR_PARSE_CHUNK_DATA:=1109,
    HTTP_REQ_ERR_PARSE_CHUNK_TRAILER:=1110
) DINT;
END_TYPE
```

7.2.2 ETcIotHttpRequestType

```
TYPE ETcIotHttpRequestType:
(
    HTTP_GET:=0,
    HTTP_POST:=1,
    HTTP_PUT:=2,
    HTTP_DELETE:=3,
    HTTP_TRACE:=4,
    HTTP_OPTIONS:=5,
    HTTP_PATCH:=6,
    HTTP_CONNECT:=7,
    HTTP_HEAD:=8
) DINT;
END_TYPE
```

7.2.3 E_IotHttpCompressionMode

```
TYPE E_IotHttpCompressionMode:
(
    NoCompression:=0,
    Deflate:=1,
    Gzip:=2
) UDINT;
END_TYPE
```

7.3 Cipher suites used

The TwinCAT IoT driver supports secure transmission using version 1.2 of the TLS standard. Up to this version, a cipher suite is a set of algorithms (key exchange, authentication, encryption, MAC) for secure transmission.

All cipher suites currently supported by the IoT driver are listed below. The information provided here refers to TwinCAT version 3.1.4024.12.

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

More Information:
www.beckhoff.com/TF6760/

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Germany
Phone: +49 5246 9630
info@beckhoff.com
www.beckhoff.com