# Table of contents

1 Foreword ........................................................................................................................................... 5  
1.1 Notes on the documentation ........................................................................................................... 5  
1.2 Safety instructions ............................................................................................................................ 6  
2 Overview ............................................................................................................................................... 7  
3 Installation ........................................................................................................................................... 8  
  3.1 System requirements ........................................................................................................................ 8  
  3.2 Installation ....................................................................................................................................... 8  
  3.3 Licensing ......................................................................................................................................... 8  
4 Technical introduction .......................................................................................................................... 11  
  4.1 HTTP/HTTPS .................................................................................................................................. 11  
  4.2 REST API ....................................................................................................................................... 12  
  4.3 Technical restrictions ......................................................................................................................... 13  
  4.4 Compression ................................................................................................................................... 13  
  4.5 Security .......................................................................................................................................... 14  
    4.5.1 Transport layer .......................................................................................................................... 14  
    4.5.2 Application level ....................................................................................................................... 18  
  4.6 Re-parameterization ........................................................................................................................ 18  
  4.7 URL redirects .................................................................................................................................. 19  
  4.8 Cookies .......................................................................................................................................... 19  
5 PLC API ............................................................................................................................................... 21  
  5.1 Tc3_IotBase .................................................................................................................................... 21  
    5.1.1 FB_IotHttpClientAwsSigV4HeaderFieldMap ............................................................................... 21  
    5.1.2 FB_IotHttpClient ....................................................................................................................... 22  
    5.1.3 FB_IotHttpRequestFieldMap ..................................................................................................... 23  
    5.1.4 FB_IotHttpRequest .................................................................................................................... 25  
    5.1.5 ST_IotSocketTls ....................................................................................................................... 30  
    5.1.6 ParameterList ............................................................................................................................ 31  
  5.2 Tc3_JsonXml ................................................................................................................................... 32  
    5.2.1 Function blocks ......................................................................................................................... 32  
    5.2.2 Interfaces .................................................................................................................................. 105  
6 Samples ............................................................................................................................................... 111  
  6.1 IotHttpSamples ............................................................................................................................... 111  
    6.1.1 OpenWeatherMap ..................................................................................................................... 111  
    6.1.2 PostmanEcho ........................................................................................................................... 113  
    6.1.3 AWS IoT .................................................................................................................................. 118  
    6.1.4 Philips Hue ............................................................................................................................... 121  
    6.1.5 Telegram .................................................................................................................................. 123  
    6.1.6 AWS Service Configuration ..................................................................................................... 127  
7 Appendix ............................................................................................................................................... 130  
  7.1 ADS Return Codes .......................................................................................................................... 130  
  7.2 Enumerations .................................................................................................................................. 134  
    7.2.1 ETcIotHttpRequestError ......................................................................................................... 134  
    7.2.2 ETcIotHttpRequestType .......................................................................................................... 135
7.2.3 E_IotHttpCompressionMode ................................................................. 135
7.3 Cipher suites used .................................................................................. 135
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®, 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:
with corresponding applications or registrations in various other countries.

EtherCAT®

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!

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="DANGER" /></td>
<td>Serious risk of injury! Failure to follow the safety instructions associated with this symbol directly endangers the life and health of persons.</td>
</tr>
<tr>
<td><img src="image" alt="WARNING" /></td>
<td>Risk of injury! Failure to follow the safety instructions associated with this symbol endangers the life and health of persons.</td>
</tr>
<tr>
<td><img src="image" alt="CAUTION" /></td>
<td>Personal injuries! Failure to follow the safety instructions associated with this symbol can lead to injuries to persons.</td>
</tr>
<tr>
<td><img src="image" alt="NOTE" /></td>
<td>Damage to the environment or devices Failure to follow the instructions associated with this symbol can lead to damage to the environment or equipment.</td>
</tr>
</tbody>
</table>

- **Tip or pointer**
  - This symbol indicates information that contributes to better understanding.
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.

![Fig. 1: TF6760](image)

**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

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

3.2 Installation

A separate 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.

5. Open the Manage Licenses tab. In the Add License column, check the check box for the license you want to add to your project (e.g. "TF6420: TC3 Database Server").

6. Open the Order Information (Runtime) tab.
   - In the tabular overview of licenses, the previously selected license is displayed with the status "missing".
7. Click **7-Day Trial License...** to activate the 7-day trial license.

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

8. Enter the code exactly as it 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.

![HTTP communication](image)

**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.

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

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

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

### 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.

<table>
<thead>
<tr>
<th>Restriction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of simultaneous HTTP requests from an HTTP client instance</td>
<td>Each instance of the function block FB_IotHttpClient 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.</td>
</tr>
<tr>
<td>Communication of large files</td>
<td>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.</td>
</tr>
</tbody>
</table>

### 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 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). The TwinCAT HTTP client notifies the REST API via a header field that both deflate and gzip are supported in receive direction.

![HTTP request with compression headers](image-url)
4.5 Security

When considering protection of data communication, a distinction can be made between two levels: protection of the transport channel (Transport layer [14]) and protection at Application level [18]. 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 [15])
- Client and server identify themselves through a certificate (see Client/Server certificate [16])
- Pre-shared keys (see Pre-shared keys [17])

Please refer to the relevant technical literature for information about the advantages and disadvantages of the different authentication types.
Exemplary explanation based on RSA

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

4.5.1.1 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 [14] is shortened as follows.
Verification of the server certificate

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

Application in TwinCAT

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

```plaintext
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 [15] chapter) is visualized in the following diagram. The individual steps of the TLS connection establishment are described in chapter Transport layer [14].
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.

```plaintext
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 [15]).

### 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.

**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.

```plaintext
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.

```c
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 Execute 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.

```text
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 PLC 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 [18]).

**Syntax**

Definition:

FUNCTION BLOCK FB_IotHttpAwsSigV4HeaderFieldMap EXTENDS FB_IotHttpHeaderFieldMap
VAR_OUTPUT
  bError : BOOL;
  hrErrorCode : HRESULT;
END_VAR

**Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>Becomes TRUE when an error situation occurs</td>
</tr>
<tr>
<td>hrErrorCode</td>
<td>HRESULT</td>
<td>Returns an ADS return code. An explanation of the possible ADS return codes can be found in the Appendix.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddField [24]</td>
<td>Adds a header field to an instance of this function block.</td>
</tr>
</tbody>
</table>

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.4024.12 or higher</td>
<td>IPC or CX (x86, x64, ARM)</td>
<td>Tc3_IotBase</td>
</tr>
</tbody>
</table>

**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

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

## Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetParameter</td>
<td>BOOL</td>
<td>Returns TRUE if the method call was successful.</td>
</tr>
</tbody>
</table>

### 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:**

```plaintext
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
```
## PLC API

### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sHostName</td>
<td>STRING (255)</td>
<td>sHostName can be specified as name or as IP address. If no information is provided, the local host is used.</td>
</tr>
<tr>
<td>nHostPort</td>
<td>UINT</td>
<td>The host port can be specified here. The default value is 80.</td>
</tr>
<tr>
<td>bKeepAlive</td>
<td>BOOL</td>
<td>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.</td>
</tr>
<tr>
<td>tConnectionTimeout</td>
<td>TIME</td>
<td>Specifies the connection timeout.</td>
</tr>
<tr>
<td>stTLS</td>
<td>ST_IotSocketTls</td>
<td>If the server offers a TLS-secured connection, the required configuration can be implemented here.</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>Becomes TRUE as soon as an error situation occurs.</td>
</tr>
<tr>
<td>hrErrorCode</td>
<td>HRESULT</td>
<td>Returns an ADS return code. An explanation of the possible ADS return codes can be found in the Appendix.</td>
</tr>
<tr>
<td>bConfigured</td>
<td>BOOL</td>
<td>Becomes TRUE when the initial configuration is finished.</td>
</tr>
<tr>
<td>bConnected</td>
<td>BOOL</td>
<td>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.</td>
</tr>
</tbody>
</table>

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disconnect</td>
<td>Disconnects the client from the server, when there is an active connection.</td>
</tr>
<tr>
<td>Execute</td>
<td>Method for background communication with the TwinCAT driver. The method must be called cyclically for every function block instance.</td>
</tr>
</tbody>
</table>

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.4024.7 or higher</td>
<td>IPC or CX (x86, x64, ARM)</td>
<td>Tc3_IotBase</td>
</tr>
</tbody>
</table>

### 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:

```plaintext
FUNCTION_BLOCK FB_IotHttpHeaderFieldMap
VAR_OUTPUT
   bError       : BOOL;
   hrErrorCode  : HRESULT;
END_VAR
```
### Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>Becomes TRUE as soon as an error situation occurs.</td>
</tr>
<tr>
<td>hrErrorCode</td>
<td>HRESULT</td>
<td>Returns an ADS return code. An explanation of the possible ADS return codes can be found in the Appendix.</td>
</tr>
</tbody>
</table>

### Methods

#### AddField

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

**Syntax**

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

#### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sField</td>
<td>STRING</td>
<td>Name of the field that is added to the header of the HTTP request.</td>
</tr>
<tr>
<td>sValue</td>
<td>STRING</td>
<td>Value of the added header field.</td>
</tr>
<tr>
<td>bAppendValue</td>
<td>BOOL</td>
<td>Replaces existing header field value (FALSE) or appends the value to the existing one (TRUE).</td>
</tr>
</tbody>
</table>

#### Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddField</td>
<td>BOOL</td>
<td>Returns TRUE when the method call was successful.</td>
</tr>
</tbody>
</table>

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.4024.7 or higher</td>
<td>IPC or CX (x86, x64, ARM)</td>
<td>Tc3_IotBase</td>
</tr>
</tbody>
</table>

### 5.1.3.1 AddField

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

**Syntax**

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

### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sField</td>
<td>STRING</td>
<td>Name of the field that is added to the header of the HTTP request.</td>
</tr>
<tr>
<td>sValue</td>
<td>STRING</td>
<td>Value of the added header field.</td>
</tr>
<tr>
<td>bAppendValue</td>
<td>BOOL</td>
<td>Replaces existing header field value (FALSE) or appends the value to the existing one (TRUE).</td>
</tr>
</tbody>
</table>

### Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddField</td>
<td>BOOL</td>
<td>Returns TRUE when the method call was successful.</td>
</tr>
</tbody>
</table>

### 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sField</td>
<td>STRING</td>
<td>Name of the field to be found in the header.</td>
</tr>
<tr>
<td>pValue</td>
<td>PVOID</td>
<td>Pointer to the data memory where the found header field is to be stored.</td>
</tr>
<tr>
<td>nValueSize</td>
<td>UDINT</td>
<td>The size of the memory variables.</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FindField</td>
<td>BOOL</td>
<td>Returns TRUE if the method call was successful.</td>
</tr>
</tbody>
</table>

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 : E_IotHttpRequestError;
  nStatusCode : UINT;
  nContentSize : UDINT;
END_VAR
### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sContentType</td>
<td>STRING</td>
<td>Content type of the request, e.g. the initial value &quot;text/html; charset=UTF-8&quot;.</td>
</tr>
<tr>
<td>eCompressionMode</td>
<td>E_IotHttpCompressionMode</td>
<td>Enumeration of the different compression modes. Further information is provided in the chapter Compression [13]. The list can be found in the Appendix.</td>
</tr>
</tbody>
</table>

### Outputs

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

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetContent [26]</td>
<td>Reads the response from the HTTP server.</td>
</tr>
<tr>
<td>GetHeaderField [27]</td>
<td>Reads the value from a specified header field.</td>
</tr>
<tr>
<td>GetJsonDomContent [28]</td>
<td>Parses the server response as JSON object.</td>
</tr>
<tr>
<td>SendJsonDomRequest [28]</td>
<td>Sends a request to the server as JSON object.</td>
</tr>
<tr>
<td>SendRequest [29]</td>
<td>Sends a HTTP command to the server.</td>
</tr>
</tbody>
</table>

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.4024.7 or higher</td>
<td>IPC or CX (x86, x64, ARM)</td>
<td>Tc3_IotBase</td>
</tr>
</tbody>
</table>

### 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pContent</td>
<td>PVOID</td>
<td>Pointer to the data memory, where the read data should be stored.</td>
</tr>
<tr>
<td>nContentSize</td>
<td>UDINT</td>
<td>Size of content payload.</td>
</tr>
<tr>
<td>bSetNullTermination</td>
<td>BOOL</td>
<td>Specifies if the payload should be read with null termination or without null termination.</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetContent</td>
<td>BOOL</td>
<td>Returns TRUE if the method call was successful.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sField</td>
<td>STRING</td>
<td>Name of the specified header field.</td>
</tr>
<tr>
<td>pValue</td>
<td>PVOID</td>
<td>Pointer to the data memory, where the read data should be stored.</td>
</tr>
<tr>
<td>nValueSize</td>
<td>UDINT</td>
<td>The maximum size that the buffer of this method can have.</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetHeaderField</td>
<td>BOOL</td>
<td>Returns TRUE if the method call was successful.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fbJson</td>
<td>REFERENCE TO FB_JsonDomParser</td>
<td>The instance of the JsonDomParser which should be used for parsing the JSON object.</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetJsonDomContent</td>
<td>SJsonValue</td>
<td>JSON root node to start parsing.</td>
</tr>
</tbody>
</table>

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

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

### Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SendJsonDomRequest</td>
<td>BOOL</td>
<td>Returns TRUE when the method call was successful.</td>
</tr>
</tbody>
</table>

#### 5.1.4.5 SendRequest

This method sends a request to a HTTP server.

**Syntax**

```pascal
METHOD SendRequest : BOOL
VAR_IN_OUT CONSTANT
  sUri : STRING;
END_VAR
VAR_INPUT
  fbClient : REFERENCE TO FB_IotHttpClient;
eRequestType : ETcIoIotHttpRequestType;
pContent : PVOID;
nContentSize : UDINT;
  fbHeader : REFERENCE TO FB_IoIotHttpHeaderFieldMap;
END_VAR
```
Inputs

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

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SendRequest</td>
<td>BOOL</td>
<td>Returns TRUE if the method call was successful.</td>
</tr>
</tbody>
</table>

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

```plaintext
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) := 'tlsv1.1';
    bNoServerCertCheck     : BOOL := FALSE;
    sPskIdentity           : STRING(255*);
    aPskKey                 : ARRAY[1..64*] OF BYTE;
    nPskKeyLen              : USINT;
  END_STRUCT
END_TYPE
```
### Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sCA</td>
<td>STRING(255)</td>
<td>Certificate of the certificate authority (CA)</td>
</tr>
<tr>
<td>sCert</td>
<td>STRING(255)</td>
<td>Client certificate to be used for authentication at the server</td>
</tr>
<tr>
<td>sKeyFile</td>
<td>STRING(255)</td>
<td>Private key of the client</td>
</tr>
<tr>
<td>sKeyPwd</td>
<td>STRING(255)</td>
<td>Password of the private key file, if applicable</td>
</tr>
<tr>
<td>sCrl</td>
<td>STRING(255)</td>
<td>Path to the certificate revocation list, which may be present in PEM or DER format</td>
</tr>
<tr>
<td>sCiphers</td>
<td>STRING(255)</td>
<td>Cipher suites to be used, specified in OpenSSL string format</td>
</tr>
<tr>
<td>sVersion</td>
<td>STRING(80)</td>
<td>TLS version to be used</td>
</tr>
<tr>
<td>bNoServerCertCheck</td>
<td>BOOL</td>
<td>Disables verification of the server certificate validity</td>
</tr>
<tr>
<td>sPskIdentity</td>
<td>STRING(255)</td>
<td>PreSharedKey identity for TLS PSK connection</td>
</tr>
<tr>
<td>aPskKey</td>
<td>ARRAY[1..64] OF BYTE</td>
<td>PreSharedKey for TLS PSK connection</td>
</tr>
<tr>
<td>nPskKeyLen</td>
<td>USINT</td>
<td>Length of the PreSharedKey in bytes</td>
</tr>
</tbody>
</table>

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 through 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:**

```plaintext
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
```
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cSizeOfHttpContentType</td>
<td>UDINT(32..10000)</td>
<td>String size definition of HTTP content type (in bytes).</td>
</tr>
<tr>
<td>cSizeOfHttpHostHostName</td>
<td>UDINT(32..10000)</td>
<td>String size definition of HTTP client hostname (in bytes).</td>
</tr>
<tr>
<td>cSizeOfHttpTlsFilePaths</td>
<td>UDINT(32..10000)</td>
<td>String size definition of HTTP TLS file path (in bytes).</td>
</tr>
<tr>
<td>cSizeOfHttpTlsKeyFile</td>
<td>UDINT(32..10000)</td>
<td>String size definition of HTTP TLS key file (in bytes).</td>
</tr>
<tr>
<td>cSizeOfHttpTlsKeyPwd</td>
<td>UDINT(32..10000)</td>
<td>String size definition of HTTP TLS private key password (in bytes).</td>
</tr>
<tr>
<td>cSizeOfHttpTlsCiphers</td>
<td>UDINT(32..10000)</td>
<td>String size definition of HTTP TLS cipher suites (in bytes).</td>
</tr>
<tr>
<td>cSizeOfHttpTlsPskIdentity</td>
<td>UDINT(32..10000)</td>
<td>String size definition of HTTP TLS PSK identity (in bytes).</td>
</tr>
<tr>
<td>cSizeOfHttpTlsPskKey</td>
<td>UDINT(32..10000)</td>
<td>String size definition of HTTP TLS PSK (in bytes).</td>
</tr>
</tbody>
</table>

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 FBJsonDynDomParser [61] and thus offers the same interface.

The two derived function blocks differ only in their internal memory management. FBJsonDomParser [32] is optimized for the fast and efficient parsing and creation of JSON documents that are only changed a little. It takes up less memory space than FBJsonDynDomParser [61] and is faster, but it consumes new memory with each change (e.g. SetObject [60]). This allocated memory is only released again by calling the method NewDocument [50].

5.2.1.1.1 AddArrayMember

This method adds an array member to a JSON object.

Syntax

```plaintext
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:

```plaintext
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.
## 5.2.1.1.6 AddDcTimeMember

This method adds a DcTime member to a JSON object.

**Syntax**

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

Sample call:

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

## 5.2.1.1.7 AddDoubleMember

This method adds a Double member to a JSON object.

**Syntax**

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

Sample call:

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

## 5.2.1.1.8 AddFileTimeMember

This method adds a FileTime member to a JSON object.

**Syntax**

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

Sample call:

```plaintext
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**

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

Sample call:

```plaintext
```
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 AddInt64Member : 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
END_VAR
```

Sample call:
```
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);
jsonMem := fbJson.AddNullMember(jsonDoc, 'TestNull');
```
PLC API

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 a 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.AddUintMember(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.
PLC API

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:
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().

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

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:

```plaintext
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".

```plaintext
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.
PLC API

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 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 must be used.

Syntax
METHOD GetJson : STRING(255)
  VAR_INPUT
    v : SJsonValue;
  END_VAR
PLC API

Sample call:

```plaintext
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**

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

Sample call:

```plaintext
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**

```plaintext
METHOD GetMaxDecimalPlaces : DINT
```

Sample call:

```plaintext
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**

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

Sample call:

```plaintext
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().
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() must be used.

Syntax

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

Sample call:

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

```plaintext
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**

```plaintext
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**

```plaintext
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**

```plaintext
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**

```plaintext
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. Following the successful loading of a file, the return value of the method jumps to TRUE for one cycle and is then automatically reset by the driver.

**Syntax**

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

Sample call:

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

5.2.1.1.68 MemberBegin

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

**Syntax**

```plaintext
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**

```plaintext
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:

```plaintext
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:

```plaintext
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:

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

<table>
<thead>
<tr>
<th>5.2.1.1.79</th>
<th><strong>PushbackHexBinaryValue</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
<td></td>
</tr>
</tbody>
</table>

**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));
```

<table>
<thead>
<tr>
<th>5.2.1.1.80</th>
<th><strong>PushbackInt64Value</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>This method appends a value of the data type Int64 to the end of an array.</td>
<td></td>
</tr>
</tbody>
</table>

**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);
```

<table>
<thead>
<tr>
<th>5.2.1.1.81</th>
<th><strong>PushbackIntValue</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>This method appends a value of the data type INT to the end of an array.</td>
<td></td>
</tr>
</tbody>
</table>

**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);
```

<table>
<thead>
<tr>
<th>5.2.1.1.82</th>
<th><strong>PushbackJsonValue</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>This method appends a JSON document to the end of an array.</td>
<td></td>
</tr>
</tbody>
</table>

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

```plaintext
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:

```json
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.

```plaintext
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**

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

Sample call:

```plaintext
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. Following the successful saving of a file, the return value of the method jumps to TRUE for one cycle and is then automatically reset by the driver.

**Syntax**

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

Sample call:

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

### 5.2.1.1.92 SetArray

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

**Syntax**

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

**Sample call:**

```plaintext
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**

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

**Sample call:**

```plaintext
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**

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

**Sample call:**

```plaintext
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**

```plaintext
METHOD SetDateTime : SJsonValue
VAR_INPUT
  v : SJsonValue;
END_VAR
```

**Sample call:**

```plaintext
jsonDoc := fbJson.ParseDocument(sExistingJsonDocument);
jsonProp := fbJson.FindMember(jsonDoc, 'property');
jsonValue := fbJson.SetDateTime(jsonProp, TRUE);
```
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:
```plaintext
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**
```plaintext
METHOD SetMaxDecimalPlaces
VAR_INPUT
  dp : DINT;
END_VAR
```

Sample call:
```plaintext
nDec := fbJson.SetMaxDecimalPlaces();
```

### 5.2.1.1.104 SetNull
This method sets the value of a property to the value NULL.

**Syntax**
```plaintext
METHOD SetNull : SJsonValue
VAR_INPUT
  v : SJsonValue;
END_VAR
```

Sample call:
```plaintext
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**
```plaintext
METHOD SetDouble : SJsonValue
VAR_INPUT
  v : SJsonValue;
  value : LREAL;
END_VAR
```

Sample call:
```plaintext
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**
```plaintext
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 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. SetObject).

Requirements

<table>
<thead>
<tr>
<th>TwinCAT version</th>
<th>Hardware</th>
<th>Libraries to be integrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1, Build 4024.7</td>
<td>x86, x64, ARM</td>
<td>Tc3_JsonXml 3.3.8.0</td>
</tr>
</tbody>
</table>
5.2.1.3  **FB_JsonSaxReader**

5.2.1.3.1  **DecodeBase64**

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

**Syntax**

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

Sample call:

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

5.2.1.3.2  **DecodeDateTime**

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

**Syntax**

METHOD DecodeDateTime : BOOL
VAR_IN_OUT CONSTANT
   sDT : STRING;
END_VAR
VAR_OUTPUT
   nDT : DATE_AND_TIME;
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 01/01/2000. If the conversion was successful the method returns TRUE, otherwise it returns FALSE.

**Syntax**

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

Sample call:

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

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

Syntax

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

Sample call:

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

5.2.1.3.6  GetLastParseResult

Syntax

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

5.2.1.3.7  IsBase64

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

Syntax

METHOD IsBase64 : BOOL
VAR_IN_OUT CONSTANT
 sBase64 : STRING;
END_VAR

Sample call:

bIsBase64 := fbJson.IsBase64('SGVsbG8gVHdpbkNBVA==');
5.2.1.3.8  **IsHexBinary**

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

**Syntax**

METHOD IsHexBinary : BOOL
VAR_IN_OUT CONSTANT
  sHex : STRING;
END_VAR

**Sample call:**

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

5.2.1.3.9  **IsISO8601TimeFormat**

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

**Syntax**

METHOD IsISO8601TimeFormat : BOOL
VAR_IN_OUT CONSTANT
  sDT : STRING;
END_VAR

**Sample call:**

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

5.2.1.3.10  **Parse**

This method starts the SAX reader parsing procedure. The JSON object to be parsed and a reference to a function block, which was derived from the interface ITCcJsonSaxHandler, 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 : ITCcJsonSaxHandler;
END_VAR

5.2.1.3.11  **ParseValues**

This method starts the SAX reader parsing procedure. The JSON object to be parsed and a reference to a function block, which was derived from the interface ITCcJsonSaxValues, 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 : ITCcJsonSaxValues;
END_VAR
5.2.1.4   FB_JsSonSaxWriter

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() \[66\].

**Syntax**

```plaintext
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() \[66\].

**Syntax**

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

Sample call:

```plaintext
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() \[66\].

**Syntax**

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

Sample call:

```plaintext
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() \[66\].

**Syntax**

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

Sample call:

```plaintext
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() [66].

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() [66].

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() [66].

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 [65], AddBool [65], AddDateTime [65], AddDcTime [65], AddDint [66], AddFileTime [66], AddHexBinary [66], AddInt [69], AddReal [69], AddNull [69], AddRawArray [69], AddRawObject [70], AddReal [70], AddString [70], AddUdint [71], AddUlint [71].

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

Sample call:
```
fbJson.AddKeyFileTime('Timestamp', dcTime);
```
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**

```plaintext
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**

```plaintext
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**

```plaintext
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.
**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() [66].

Syntax

METHOD AddLint
VAR_INPUT
    value : LINT;
END_VAR

Sample call:

```plaintext
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() [66].

Syntax

METHOD AddLreal
VAR_INPUT
    value : LREAL;
END_VAR

Sample call:

```plaintext
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() [66].

Syntax

METHOD AddNull

Sample call:

```plaintext
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() [66], StartArray() [73] or as the first call after a ResetDocument() [72].
### AddRawArray

**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');
```

### 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() or StartArray() or as the first call after a ResetDocument().

**Syntax**

METHOD AddRawObject
VAR_IN_OUT CONSTANT
  rawJson : STRING;
END_VAR

**Sample call:**

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

### 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().

**Syntax**

METHOD AddReal
VAR_INPUT
  value : REAL;
END_VAR

**Sample call:**

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

### 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().

**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() [66].

**Syntax**

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

Sample call:
```
fbJson.AddKey('PropertyName');
fbJson.AddUdint(42);
```

5.2.1.4.25 AddUluint

This method adds a value of the data type ULINT to a property. Usually, a corresponding property was created beforehand with the method AddKey() [66].

**Syntax**

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

Sample call:
```
fbJson.AddKey('PropertyName');
fbJson.AddUluint(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
```

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()
```

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

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 () must be used.

Syntax

METHOD GetDocument : STRING(255)

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

Sample call:
nLength := fbJson.GetDocumentLength();

5.2.1.4.31 GetMaxDecimalPlaces

5.2.1.4.32 ResetDocument

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

Syntax

METHOD ResetDocument

Sample call:
fbJson.ResetDocument();
5.2.1.4.33 SetMaxDecimalPlaces

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()

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

Sample call:

fbJson.StartObject();

5.2.1.5 FB_JsonReadWriteDataType

5.2.1.5.1 AddJsonKeyPropertiesFromSymbol

With the aid of this method, metadata can be added via PLC attributes to the JSON representation of a PLC data structure on an FB_JsonSaxWriter 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 : FB_JsonSaxWriter;
END_VAR
VAR_IN_OUT CONSTANT
sKey : STRING;
sDatatype : STRING;
sProperties : STRING;
END_VAR

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

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

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

Sample call:

fbJsonSaxWriter.ResetDocument()
fbJsonDataType.AddJsonKeyPropertiesFromSymbol(fbJsonSaxWriter, 'MetaData','ST_Values','Unit|DisplayName');
5.2.1.5.2 AddJsonKey{ValueFromSymbol}

This method generates the JSON representation of a PLC data structure on an FB_JsonSaxWriter 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, 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 AddJsonKey{ValueFromSymbol : BOOL
VAR_IN_OUT
  fbWriter : FB_JsonSaxWriter;
END_VAR
VAR_IN_OUT CONSTANT
  sDatatype : STRING;
END_VAR
VAR_INPUT
  nData : UDINT;
pData : PVOID;
END_VAR

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

Sample call:

fbJsonSaxWriter.ResetDocument()
fbJsonDataType.AddJsonKey{ValueFromSymbol(fbJsonSaxWriter, 'Values','ST_Values',SIZEOF(stValues), ADR(stValues));

5.2.1.5.3 AddJsonValueFromSymbol

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

Syntax

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

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

Sample call:

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

5.2.1.5.4 CopyJsonStringFromSymbol

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

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

Sample call:

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

5.2.1.5.5 CopyJsonStringFromSymbolProperties

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

Sample call:

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

5.2.1.5.6 GetDataTypeNameByAddress

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

Syntax

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

Sample call:

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

5.2.1.5.7 GetJsonFromSymbol

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

Syntax

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

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

Sample call:

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

### 5.2.1.5.8 GetJsonStringFromSymbol

This method generates the corresponding JSON representation of a symbol. In contrast to the `AddJsonValueFromSymbol()` [74] 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` [74] must be used.

Syntax

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

Sample call:

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

### 5.2.1.5.9 GetJsonStringFromSymbolProperties

This method generates a corresponding JSON representation of PLC attributes on a symbol. In contrast to the `AddJsonKeyPropertiesFromSymbol()` [73] 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` [75] must be used.
Syntax
METHOD GetJsonStringFromSymbolProperties : STRING(255)
VAR_IN_OUT CONSTANT
  sDatatype : STRING;
  sProperties : STRING;
END_VAR

Sample call:

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

5.2.1.5.10 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

Sample call:

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

5.2.1.5.11 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

Sample call:

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

5.2.1.5.12 GetSymbolNameByAddress

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

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

Sample call:

sBuffer := fbJsonDataType.GetSymbolNameByAddress(SIZEOF(stValues), ADR(stValues));
5.2.1.5.13 **SetSymbolFromJson**

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

**Syntax**

```
METHOD SetSymbolFromJson : BOOL
VAR_IN_OUT CONSTANT
   sJson : STRING;
   sDatatype : STRING;
END_VAR
VAR_INPUT
   nData : UDINT;
   pData : PVOID;
END_VAR
Sample call:
fbJsonDataType.SetSymbolFromJson(sJson,'ST_Values',SIZEOF(stValuesReceive), ADR(stValuesReceive));
```

5.2.1.6 **FB_XmlDomParser**

5.2.1.6.1 **AppendAttribute**

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

**Syntax**

```
METHOD AppendAttribute : SXmlAttribute
VAR_INPUT
   n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
   name : STRING;
   value : STRING;
END_VAR
Sample call:
objAttribute := fbXml.AppendAttribute(objMachine, 'Name', 'some value');
```

5.2.1.6.2 **AppendAttributeAsBool**

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

**Syntax**

```
METHOD AppendAttributeAsBool : SXmlAttribute
VAR_INPUT
   n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
   name : STRING;
END_VAR
VAR_INPUT
   value : BOOL;
END_VAR
Sample call:
objAttribute := fbXml.AppendAttributeAsBool(objMachine, 'Name', TRUE);
```
5.2.1.6.3 AppendAttributeAsDouble

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

**Syntax**

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

Sample call:

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

5.2.1.6.4 AppendAttributeAsFloat

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

**Syntax**

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

Sample call:

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

5.2.1.6.5 AppendAttributeAsInt

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

**Syntax**

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

Sample call:

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

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

Syntax

```plaintext
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:

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

5.2.1.6.7 AppendAttributeAsUint

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

Syntax

```plaintext
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:

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

5.2.1.6.8 AppendAttributeAsUlint

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

Syntax

```plaintext
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:

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

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

Syntax

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

Sample call:

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

5.2.1.6.10 AppendChild

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

Syntax

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

Sample call:

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

5.2.1.6.11 AppendChildAsBool

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

Syntax

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

Sample call:

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

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

Syntax

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

Sample call:

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

5.2.1.6.13 AppendChildAsFloat

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

Syntax

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

Sample call:

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

5.2.1.6.14 AppendChildAsInt

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

Syntax

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

Sample call:

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

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

**Syntax**

METHOD AppendChildAsLint : SXmlNode
VAR_INPUT
  n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name : STRING;
END_VAR
VAR_INPUT
  value : LINT;
END_VAR
Sample call:
xmlNewNode := fbXml.AppendChildAsLint(xmlExisting, 'SomeName', 42);

5.2.1.6.16  AppendChildAsUint

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

**Syntax**

METHOD AppendChildAsUint : SXmlNode
VAR_INPUT
  n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name : STRING;
END_VAR
VAR_INPUT
  value : UDINT;
END_VAR
Sample call:
xmlNewNode := fbXml.AppendChildAsUint(xmlExisting, 'SomeName', 42);

5.2.1.6.17  AppendChildAsUlint

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

**Syntax**

METHOD AppendChildAsUlint : SXmlNode
VAR_INPUT
  n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name : STRING;
END_VAR
VAR_INPUT
  value : ULINT;
END_VAR
Sample call:
xmlNewNode := fbXml.AppendChildAsUlint(xmlExisting, 'SomeName', 42);
5.2.1.6.18 AppendCopy

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

**Syntax**

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

Sample call:

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

5.2.1.6.19 AppendNode

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

**Syntax**

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

Sample call:

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

5.2.1.6.20 Attributes

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

**Syntax**

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

Sample call:

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

5.2.1.6.21 AttributeAsBool

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

**Syntax**

METHOD AttributeAsBool : BOOL
VAR_INPUT
 a : SXmlAttribute;
END_VAR
Sample call:
bValue := fbXml.AttributeAsBool(xmlAttr);

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

**Syntax**

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

Sample call:
lrValue := fbXml.AttributeAsDouble(xmlAttr);

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

**Syntax**

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

Sample call:
rValue := fbXml.AttributeAsFloat(xmlAttr);

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

**Syntax**

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

Sample call:
nValue := fbXml.AttributeAsInt(xmlAttr);

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

**Syntax**

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

Sample call:
nValue := fbXml.AttributeAsLint(xmlAttr);
5.2.1.6.26  AttributeAsUint

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

Syntax

METHOD AttributeAsUint : UDINT
VAR_INPUT
  a : SXmlAttribute;
END_VAR

Sample call:

nValue := fbXml.AttributeAsUint(xmlAttr);

5.2.1.6.27  AttributeAsUlint

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

Syntax

METHOD AttributeAsUlint : ULINT
VAR_INPUT
  a : SXmlAttribute;
END_VAR

Sample call:

nValue := fbXml.AttributeAsUlint(xmlAttr);

5.2.1.6.28  AttributeBegin

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

Syntax

METHOD AttributeBegin : SXmlIterator
VAR_INPUT
  n : SXmlNode;
END_VAR

Sample call:

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

5.2.1.6.29  AttributeFromlterator

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

Syntax

METHOD AttributeFromIterator : SXmlAttribute
VAR_INPUT
  it : SXmlIterator;
END_VAR

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

5.2.1.6.30   AttributeName

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

Syntax

METHOD AttributeName : STRING
VAR_INPUT
    a : SXmlAttribute;
END_VAR

Sample call:

sName := fbXml.AttributeName(xmlAttr);

5.2.1.6.31   Attributes

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

Syntax

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

Sample call:

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

5.2.1.6.32   AttributeText

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

Syntax

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

Sample call:

sText := fbXml.AttributeText(xmlAttr);

5.2.1.6.33   Begin

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

METHOD Begin : SXmlIterator
VAR_INPUT
  n : SXmlNode;
END_VAR

Sample call:

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

5.2.1.6.34  BeginByName

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

Syntax

METHOD BeginByName : SXmlIterator
VAR_INPUT
  n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name : STRING;
END_VAR

Sample call:

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

5.2.1.6.35  Child

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

Syntax

METHOD ChildByName : SXmlNode
VAR_INPUT
  n : SXmlNode;
END_VAR

Sample call:

xmlChild := fbXml.Child(xmlNode);

5.2.1.6.36  ChildByAttribute

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

Syntax

METHOD ChildByAttribute : SXmlNode
VAR_INPUT
  n : SXmlNode;
VAR_IN_OUT CONSTANT
5.2.1.6.37 ChildByAttributeAndName

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

Syntax

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

Sample call:

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

5.2.1.6.38 ChildByName

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

Syntax

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

Sample call:

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

5.2.1.6.39 Children

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

Syntax

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

Sample call:

xmlRet := fbXml.Children(xmlNode, xmlIterator);
WHILE NOT fbXml.IsEnd(xmlIterator) DO
    xmlMachineNodeRef := fbXml.Node(xmlIterator);
5.2.1.6.40  ChildrenByName

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

Syntax

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

Sample call:

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

5.2.1.6.41  Compare

This method checks two iterators for equality.

Syntax

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

Sample call:

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

5.2.1.6.42  CopyAttributeText

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

Syntax

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

Sample call:

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

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

Syntax

```plaintext
METHOD CopyDocument : UDINT
  VAR_IN_OUT CONSTANT
    sXml : STRING;
  END_VAR
  VAR_INPUT
    nXml : UDINT;
  END_VAR
  Sample call:
  nLength := fbXml.CopyDocument(sTarget, SIZEOF(sTarget));
```

5.2.1.6.44 CopyNodeText

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

Syntax

```plaintext
METHOD CopyNodeText : UDINT
  VAR_INPUT
    n : SXmlNode;
  END_VAR
  VAR_IN_OUT CONSTANT
    sXml : STRING;
  END_VAR
  VAR_INPUT
    nXml : UDINT;
  END_VAR
  Sample call:
  nLength := fbXml.CopyNodeText(xmlNode, sTarget, SIZEOF(sTarget));
```

5.2.1.6.45 CopyNodeXml

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

Syntax

```plaintext
METHOD CopyNodeXml : UDINT
  VAR_INPUT
    a : SXmlNode;
  END_VAR
  VAR_IN_OUT CONSTANT
    sXml : STRING;
  END_VAR
  VAR_INPUT
    nXml : UDINT;
  END_VAR
  Sample call:
  nLength := fbXml.CopyNodeXml(xmlNode, sTarget, SIZEOF(sTarget));
```
5.2.1.6.46 FirstNodeByPath

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

**Syntax**

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

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

5.2.1.6.47 GetAttributeTextLength

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

**Syntax**

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

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

5.2.1.6.48 GetDocumentLength

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

**Syntax**

```
METHOD GetDocumentLength : UDINT
```

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

5.2.1.6.49 GetDocumentNode

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

**Syntax**

```
METHOD GetDocumentNode : SXmlNode
```

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

5.2.1.6.50 GetNodeTextLength

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

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

Syntax

METHOD GetNodeXmlLength : UDINT
VAR_INPUT
  n : SXmlNode;
END_VAR

Sample call:

nLength := fbXml.GetNodeXmlLength(xmlNode);

5.2.1.6.52 GetRootNode

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

Syntax

METHOD GetRootNode : SXmlNode

Sample call:

xmlRootNode := fbXml.GetRootNode();

5.2.1.6.53 InsertAttributeCopy

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

Syntax

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

Sample call:

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

5.2.1.6.54 InsertAttribute

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

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

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

---

**5.2.1.6.55  InsertChild**

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

**Syntax**

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

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

---

**5.2.1.6.56  InsertCopy**

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

**Syntax**

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

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

---

**5.2.1.6.57  IsEnd**

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

**Syntax**

METHOD IsEnd : BOOL
VAR_INPUT
  it : SXmlIterator;
END_VAR

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

### 5.2.1.6.58 LoadDocumentFromFile

This method loads an XML document from a file. The absolute path to the file is transferred to the method as input parameter. A rising edge on the input parameter bExec triggers the loading procedure. Following the successful loading of a file, the return value of the method jumps to TRUE for one cycle and is then automatically reset by the driver.

**Syntax**

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

Sample call:

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

### 5.2.1.6.59 NewDocument

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

**Syntax**

METHOD NewDocument : BOOL

Sample call:

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

### 5.2.1.6.60 Next

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

**Syntax**

METHOD Next : SXmlIterator
VAR_INPUT
    it : SXmlIterator;
END_VAR

Sample call:

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

### 5.2.1.6.61 NextAttribute

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

METHOD NextAttribute : SXmlAttribute
VAR_INPUT
  a : SXmlAttribute;
END_VAR

Sample call:
xmlNextAttr := fbXml.NextAttribute(xmlAttr);

5.2.1.6.62 NextByName

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

Syntax

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

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

5.2.1.6.63 NextSibling

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

Syntax

METHOD NextSibling : SXmlNode
VAR_INPUT
  n : SXmlNode;
END_VAR

Sample call:
xmlSibling := fbXml.NextSibling(xmlNode);

5.2.1.6.64 NextSiblingByName

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

Syntax

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

Sample call:
xmlSibling := fbXml.NextSibling(xmlNode, 'SomeName');
5.2.1.6.65  Node

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

Syntax

METHOD Node : SXmlNode
VAR_INPUT
  it : SXmlIterator;
END_VAR

Sample call:

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

5.2.1.6.66  NodeAsBool

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

Syntax

METHOD NodeAsBool : BOOL
VAR_INPUT
  n : SXmlNode;
END_VAR

Sample call:

bXmlNode := fbXml.NodeAsBool(xmlMachine1);

5.2.1.6.67  NodeAsDouble

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

Syntax

METHOD NodeAsDouble : LREAL
VAR_INPUT
  n : SXmlNode;
END_VAR

Sample call:

lrXmlNode := fbXml.NodeAsDouble(xmlMachine1);

5.2.1.6.68  NodeAsFloat

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

Syntax

METHOD NodeAsFloat : REAL
VAR_INPUT
  n : SXmlNode;
END_VAR

Sample call:

rXmlNode := fbXml.NodeAsFloat(xmlMachine1);
5.2.1.6.69  **NodeAsInt**

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

**Syntax**

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

Sample call:

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

5.2.1.6.70  **NodeAsLint**

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

**Syntax**

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

Sample call:

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

5.2.1.6.71  **NodeAsUint**

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

**Syntax**

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

Sample call:

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

5.2.1.6.72  **NodeAsUlint**

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

**Syntax**

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

Sample call:

nXmlNode:= fbXml.NodeAsUlint(xmlMachine1);
```
5.2.1.6.73  **NodeName**

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

**Syntax**

METHOD NodeName : STRING
VAR_INPUT
  n : SXmlNode;
END_VAR

Sample call:

sNodeName := fbXml.NodeName(xmlMachine1);

5.2.1.6.74  **NodeText**

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

**Syntax**

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

Sample call:

sMachine1Name := fbXml.NodeText(xmlMachine1);

5.2.1.6.75  **ParseDocument**

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

**Syntax**

METHOD ParseDocument : SXmlNode
VAR_IN_OUT CONSTANT
  sXml : STRING;
END_VAR

Sample call:

xmlDoc := fbXml.ParseDocument(sXmlToParse);

5.2.1.6.76  **RemoveChild**

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

**Syntax**

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

Sample call:

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

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

Syntax

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

Sample call:

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

5.2.1.6.78  SaveDocumentToFile

This method saves the current XML document in a file. The absolute path to the file is transferred to the method as input parameter. A rising edge at the input parameter bExec triggers the saving procedure. Following the successful saving of a file, the return value of the method jumps to TRUE for one cycle and is then automatically reset by the driver.

Syntax

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

Sample call:

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

5.2.1.6.79  SetAttribute

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

Syntax

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

Sample call:

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

5.2.1.6.80  SetAttributeAsBool

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

METHOD SetAttributeAsBool : SXmlAttribute
VAR_INPUT
  a : SXmlAttribute;
  value : BOOL;
END_VAR

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

5.2.1.6.81  SetAttributeAsDouble

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

**Syntax**

METHOD SetAttributeAsDouble : SXmlAttribute
VAR_INPUT
  a : SXmlAttribute;
  value : LREAL;
END_VAR

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

5.2.1.6.82  SetAttributeAsFloat

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

**Syntax**

METHOD SetAttributeAsFloat : SXmlAttribute
VAR_INPUT
  a : SXmlAttribute;
  value : REAL;
END_VAR

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

5.2.1.6.83  SetAttributeAsInt

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

**Syntax**

METHOD SetAttributeAsInt : SXmlAttribute
VAR_INPUT
  a : SXmlAttribute;
  value : DINT;
END_VAR

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

5.2.1.6.84  SetAttributeAsLint

This method sets the value of an attribute. The value has the data type Integer64.
**Syntax**

METHOD SetAttributeAsLint : SXmlAttribute  
VAR_INPUT  
   a : SXmlAttribute;  
   value : LINT;  
END_VAR

Sample call:

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

5.2.1.6.85 **SetAttributeAsUint**

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

**Syntax**

METHOD SetAttributeAsUint : SXmlAttribute  
VAR_INPUT  
   a : SXmlAttribute;  
   value : UDINT;  
END_VAR

Sample call:

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

5.2.1.6.86 **SetAttributeAsUlint**

This method sets the value of an attribute. The value has the data type Unsigned Integer64.

**Syntax**

METHOD SetAttributeAsUlint : SXmlAttribute  
VAR_INPUT  
   a : SXmlAttribute;  
   value : ULINT;  
END_VAR

Sample call:

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

5.2.1.6.87 **SetChild**

This method sets the value of an XML node. The value is transferred to the method as input parameter of data type String. The input parameter cdata indicates whether the value of the node is to be encapsulated in a CDATA function block, so that certain special characters such as "<" and ">" are allowed as values.

**Syntax**

METHOD SetChild : SXmlNode  
VAR_INPUT  
   n : SXmlNode;  
END_VAR  
VAR_IN_OUT CONSTANT  
   value : STRING;  
END_VAR  
VAR_INPUT  
   cdata : BOOL;  
END_VAR

Sample call:

XmlNode := fbXml.SetChild(xmlExistingNode, 'SomeText', FALSE);
5.2.1.6.88  SetChildAsBool

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

**Syntax**

```plaintext
METHOD SetChildAsBool : SXmlNode
VAR_INPUT
    n : SXmlNode;
    value : BOOL;
END_VAR
```

Sample call:

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

5.2.1.6.89  SetChildAsDouble

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

**Syntax**

```plaintext
METHOD SetChildAsDouble : SXmlNode
VAR_INPUT
    n : SXmlNode;
    value : LREAL;
END_VAR
```

Sample call:

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

5.2.1.6.90  SetChildAsFloat

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

**Syntax**

```plaintext
METHOD SetChildAsFloat : SXmlNode
VAR_INPUT
    n : SXmlNode;
    value : REAL;
END_VAR
```

Sample call:

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

5.2.1.6.91  SetChildAsInt

This method sets the value of an XML node. The value is transferred to the method as input parameter of data type Integer.

**Syntax**

```plaintext
METHOD SetChildAsInt : SXmlNode
VAR_INPUT
    n : SXmlNode;
    value : DINT;
END_VAR
```

Sample call:

```plaintext
xmlNode := fbXml.SetChildAsInt(xmlExistingNode, 42);
```
5.2.1.6.92 SetChildAsLint

This method sets the value of an XML node. The value is transferred to the method as input parameter of data type Integer64.

Syntax

METHOD SetChildAsLint : SXmlNode
VAR_INPUT
   n : SXmlNode;
   value : LINT;
END_VAR

Sample call:
xmlNode := fbXml.SetChildAsLint(xmlExistingNode, 42);

5.2.1.6.93 SetChildAsUint

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

Syntax

METHOD SetChildAsUint : SXmlNode
VAR_INPUT
   n : SXmlNode;
   value : UDINT;
END_VAR

Sample call:
xmlNode := fbXml.SetChildAsUint(xmlExistingNode, 42);

5.2.1.6.94 SetChildAsUlint

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

Syntax

METHOD SetChildAsUlint : SXmlNode
VAR_INPUT
   n : SXmlNode;
   value : ULINT;
END_VAR

Sample call:
xmlNode := fbXml.SetChildAsUlint(xmlExistingNode, 42);

5.2.1.7 FB_JwtEncode

The function block enables the creation and signing of a JSON Web Token (JWT).

Syntax

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_FUNCTION_BLOCK
END_VAR
VAR_IN_OUT CONSTANT
  sJwt   : STRING;
END_VAR
VAR_OUTPUT
  bBusy  : BOOL;
  bError : BOOL;
  hrErrorCode : HRESULT;
  initStatus : HRESULT;
END_VAR

### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bExecute</td>
<td>BOOL</td>
<td>A rising edge activates processing of the function block.</td>
</tr>
<tr>
<td>sHeaderAlg</td>
<td>STRING(46)</td>
<td>The algorithm to be used for the JWT header, e.g. RS256.</td>
</tr>
<tr>
<td>sPayload</td>
<td>STRING(1023)</td>
<td>The JWT payload to be used.</td>
</tr>
<tr>
<td>sKeyFilePath</td>
<td>STRING(511)</td>
<td>Path to the private key to be used for the signature of the JWT.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>ADS timeout, which is used internally for file access to the private key.</td>
</tr>
<tr>
<td>pKey</td>
<td>PVVOID</td>
<td>Buffer for the private key to be read.</td>
</tr>
<tr>
<td>nKeySize</td>
<td>UDINT</td>
<td>Maximum size of the buffer.</td>
</tr>
<tr>
<td>sJwt</td>
<td>STRING</td>
<td>Contains the fully coded and signed JWT after the function block has been processed.</td>
</tr>
<tr>
<td>nJwtSize</td>
<td>UDINT</td>
<td>Size of the generated JWT including zero termination.</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>Is TRUE as long as processing of the function block is in progress.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>Becomes TRUE as soon as an error situation occurs.</td>
</tr>
<tr>
<td>hrErrorCode</td>
<td>HRESULT</td>
<td>Returns an error code if the bError output is set. An explanation of the possible error codes can be found in the Appendix.</td>
</tr>
</tbody>
</table>

### Requirements

<table>
<thead>
<tr>
<th>TwinCAT version</th>
<th>Hardware</th>
<th>Libraries to be integrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT 3.1, Build 4024.4</td>
<td>x86, x64, ARM</td>
<td>Tc3_JsonXml 3.3.6.0</td>
</tr>
</tbody>
</table>

### 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.
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

\[\text{value} : \text{LINT};\]

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

\[\text{value} : \text{LREAL};\]

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

\[\text{value} : \text{STRING};\]

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
5.2.2.3  **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**

```plaintext
METHOD OnLintValue : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
    value : LINT;
END_VAR
```

5.2.2.4  **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**

```plaintext
METHOD OnLrealValue : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
    value : LREAL;
END_VAR
```

5.2.2.5  **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**

```plaintext
METHOD OnNull : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
END_VAR
```

5.2.2.6  **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**

```plaintext
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.

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.

```plaintext
PROGRAM MAIN
VAR
// trigger command execution for OpenWeatherMap samples
bGetOpenWeatherMap : BOOL;

fbHttpClientOpenWeatherMap : FB_IotHttpClient := (sHostName := 'api.openweathermap.org',
```
bKeepAlive:=TRUE, tConnectionTimeout:=T#10S);

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
    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
          sResultValue:= fbRequest.GetJson(sResultValue);
          nValidResCount:= nValidResCount+1;
        END_IF
      END_IF
    END_IF
  END_VAR
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_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 authentication.

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;
END_VAR
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:= E TclHttpRequestType.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
        END_IF
      END_IF
      nResCount:= nResCount+1;
    END_IF
    nState:= 0;
    bBusy:= FALSE;
    IF bError THEN
      nErrCount:= nErrCount+1;
    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 webservice is parsed. The JSON response contains the nReqCount variable that has been send to the webservice.

```plaintext
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;  
 nRequestCount : 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:= ETC_IotHttpRequestType.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.HasValue(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
```
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;
  nReqCount : UDINT;
  nResCount : UDINT;
  nValidResCount : UDINT;
  nErrCount : UDINT;
  nGetContentResult : BOOL;
  sContent : STRING(511);
  nGetJsonResult : BOOL;
  jsonDoc : SJsonValue;
  jsonVal : SJsonValue;
  sResultValue : STRING;
  nState := 1;
  nReqCount := nReqCount + 1;
  nValidResCount := nValidResCount + 1;
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
            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;
              END_IF
            END_IF
          END_IF
        END_IF
      END_IF
    END_IF
## 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 webservice is parsed. The JSON response contains the information if the authentication has been successful.

```plaintext
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
   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
  END_IF
END_IF
nResCount:= nResCount+1;
END_IF
nState:= 0;
bBusy:= FALSE;
IF bError THEN
  nErrCount:= nErrCount+1;
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.

```plaintext
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:= EToIotHttpRequestType.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
          nValidResCount:= nValidResCount+1;
          bError:= FALSE;
        END_IF
        nResCount:= nResCount+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.

```plaintext
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
        END_IF
        nResCount:= nResCount+1;
      END_IF
    END_IF
END_VAR
```
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.

![Philips Hue Bulbs](image)

**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/](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.

```tcf
PROGRAM MAIN
VAR
  // trigger command execution for Philips Hue samples
  bPutPhilipsHue : BOOL;
  fbHttpClientPhilipsHue : 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
```

---

**Samples**

| nState:= 0; |
| bBusy:= FALSE; |
| IF bError THEN |
| nErrCount:= nErrCount+1; |
| 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.

![Philips Hue Bulbs](image)

**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/](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.

```tcf
PROGRAM MAIN
VAR
  // trigger command execution for Philips Hue samples
  bPutPhilipsHue : BOOL;
  fbHttpClientPhilipsHue := (sHostName='172.17.x.x',
  bKeepAlive=TRUE, tConnectionTimeout=T#10S);
  fbHttpPutPhilipsHue;
  bBlinkingMode : BOOL;
  fbTimer := (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
    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
  fbHttpClient : 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/qiyut7ubQQQv2xKJfFe9cVvIo0UGHtIk2eYsIfGX/lights/1/state', fbClient:=fbClient, eRequestType:=ETcIoHttpType.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;
  END_IF
  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;
      bGetJsonResult:= FALSE;
      jsonDoc:= fbRequest.GetJsonDomContent(fbJson);
      IF jsonDoc <> 0 THEN
        // do something with status response
        nValidResCount:= nValidResCount+1;
        bError:= FALSE;
      END_IF
    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
/newbot

BotFather
Alright, a new bot. How are we going to call it? Please choose a name for your bot.

Michael
BeckhoffWebinar

BotFather
Good. Now let's choose a username for your bot. It must end in `bot`. Like this, for example: TetrisBot or tetris_bot.

MK
BeckhoffBotBot

BotFather
Done! Congratulations on your new bot. You will find it at t.me/BeckhoffBotBot. 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:

1106704352:AAExFedaf7f9CojJ19

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.

```
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
```

**Fig. 6: Telegram Chat**

<table>
<thead>
<tr>
<th>Time</th>
<th>User</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>6:36:15 PM</td>
<td>Michael</td>
<td>/start</td>
</tr>
<tr>
<td>7:36:22 PM</td>
<td>Michael</td>
<td>Message for retrieving Chat ID</td>
</tr>
<tr>
<td>7:38:59 PM</td>
<td>BeckhoffWebinar</td>
<td>This is a message out of TwinCAT 3</td>
</tr>
<tr>
<td>3:27:32 PM</td>
<td>BeckhoffWebinar</td>
<td>TwinCAT Webinar Message</td>
</tr>
<tr>
<td>5:27:03 PM</td>
<td>BeckhoffWebinar</td>
<td>English Webinar slot</td>
</tr>
</tbody>
</table>

Write a message...
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.

```plaintext
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);
    jsonDoc : SJsonValue;
    jsonVal : SJsonValue;
    sResultValue : STRING;
    nReqCount : UDINT;
    nResCount : UDINT;
    nValidResCount : UDINT;
    nErrCount : UDINT;
    fbFormatString : FB_FormatString;
    sBotApiKey : STRING(500) := '1106704362:AAExFeda7Jf9CojjI9whLEzPeE4KDLzEnf';
    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:=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
```

```plaintext
```
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_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 [18] 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 [19], 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  
fHttpClientAWSSigV4.bConfigured:=TRUE;  
fbHttpClientAWSSigV4.nPort:=443;  
fbHttpClientAWSSigV4.stTLS.bNoServerCertCheck:=TRUE;  
END_IF  
IF fbHttpClientAWSSigV4.bConfigured THEN  
fbHttpGetAWSSigV4(bSend:= bGetAWSSigV4, fbClient:= fbHttpClientAWSSigV4);  
END_IF  
fHttpClientAWSSigV4.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**

<table>
<thead>
<tr>
<th>bSend</th>
<th>BOOL;</th>
</tr>
</thead>
<tbody>
<tr>
<td>END_VAR</td>
<td></td>
</tr>
</tbody>
</table>

**VAR_IN_OUT**

<table>
<thead>
<tr>
<th>fbClient</th>
<th>FB_IotHttpClient;</th>
</tr>
</thead>
<tbody>
<tr>
<td>END_VAR</td>
<td></td>
</tr>
</tbody>
</table>

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>bBusy</th>
<th>BOOL;</th>
</tr>
</thead>
<tbody>
<tr>
<td>bError</td>
<td>BOOL;</td>
</tr>
<tr>
<td>END_VAR</td>
<td></td>
</tr>
</tbody>
</table>

**VAR**

<table>
<thead>
<tr>
<th>fbRequest</th>
<th>FB_IotHttpRequest;</th>
</tr>
</thead>
<tbody>
<tr>
<td>fbSig4Header</td>
<td>FB_IotHttpAwsSigV4HeaderFieldMap;</td>
</tr>
<tr>
<td>nState</td>
<td>UDINT;</td>
</tr>
<tr>
<td>RisingEdge</td>
<td>R_TRIG;</td>
</tr>
<tr>
<td>nReqCount</td>
<td>UDINT;</td>
</tr>
<tr>
<td>nResCount</td>
<td>UDINT;</td>
</tr>
<tr>
<td>nValidResCount</td>
<td>UDINT;</td>
</tr>
<tr>
<td>nErrCount</td>
<td>UDINT;</td>
</tr>
<tr>
<td>RequestUrl</td>
<td>STRING(500):='?Action=RunInstances&amp;ImageId=ami-0229f7666f517b31e&amp;InstanceType=t2.small&amp;KeyName=TestKDB&amp;MaxCount=1&amp;MinCount=1&amp;Version=2016-11-15';</td>
</tr>
<tr>
<td>sRequestUrl</td>
<td>STRING(500):='?Action=DescribeInstances&amp;Version=2016-11-15';</td>
</tr>
<tr>
<td>sService</td>
<td>STRING:='ec2';</td>
</tr>
<tr>
<td>sRegion</td>
<td>STRING:='us-east-1';</td>
</tr>
<tr>
<td>sAccessKey</td>
<td>STRING:='censored';</td>
</tr>
<tr>
<td>sSecretKey</td>
<td>STRING:='censored';</td>
</tr>
<tr>
<td>sSignedHeaders</td>
<td>STRING:='host;x-amz-date';</td>
</tr>
<tr>
<td>bSetParameter</td>
<td>BOOL;</td>
</tr>
<tr>
<td>END_VAR</td>
<td></td>
</tr>
</tbody>
</table>

**RisingEdge** (CLK:= bSend);

**CASE nState OF**

0:

**IF RisingEdge.Q THEN**

bSetParameter:=fbSig4Header.setParameter(sService, sRegion, sAccessKey, sSecretKey, sSignedHeaders);

**IF fbRequest.sendRequest(sUri:=sConMessage, fbClient:=fbClient, eRequestType:=ETcIotHttpRequestType.HTTP_Get, 0, 0, fbHeader:=fbSig4Header) THEN**

nState:= 1;

**END_IF**

**ELSE**

**IF NOT fbRequest.bBusy THEN**

bError:= TRUE;

**END_IF**

**IF NOT fbRequest.bError THEN**

bSetParameter:=FALSE;

**END_IF**

1:

**IF NOT fbRequest.bBusy THEN**

bError:= TRUE;

**IF NOT fbRequest.bError THEN**

bSetContentResult:= fbRequest.GetContent(pContent:=ADR(sContent), nContentSize:= SIZEOF (sContent)), bSetNullTermination:= TRUE);

**IF fbRequest.nStatusCode >= 200 AND fbRequest.nStatusCode < 300 THEN**

nResCount:= nResCount+1;

**ELSE**

**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: 0x000... 0x130..., 0x500... 0x131..., 0x700... 0x133...

<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
<th>HRESULT</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>0</td>
<td>0x9811 0000</td>
<td>ERR_NOERROR</td>
<td>No error.</td>
</tr>
<tr>
<td>0x1</td>
<td>1</td>
<td>0x9811 0001</td>
<td>ERR_INTERNAL</td>
<td>Internal error.</td>
</tr>
<tr>
<td>0x2</td>
<td>2</td>
<td>0x9811 0002</td>
<td>ERR_NORTIME</td>
<td>No real-time.</td>
</tr>
<tr>
<td>0x3</td>
<td>3</td>
<td>0x9811 0003</td>
<td>ERR_ALLOCCLOCKEDMEM</td>
<td>Allocation locked – memory error.</td>
</tr>
<tr>
<td>0x4</td>
<td>4</td>
<td>0x9811 0004</td>
<td>ERR_INSERTMAILBOX</td>
<td>Mailbox full – the ADS message could not be sent. Reducing the number of ADS messages per cycle will help.</td>
</tr>
<tr>
<td>0x5</td>
<td>5</td>
<td>0x9811 0005</td>
<td>ERR_WRONGRECEIVEHMSG</td>
<td>Wrong HMSG.</td>
</tr>
<tr>
<td>0x6</td>
<td>6</td>
<td>0x9811 0006</td>
<td>ERR_TARGETPORTNOTFOUND</td>
<td>Target port not found – ADS server is not started or is not reachable.</td>
</tr>
<tr>
<td>0x7</td>
<td>7</td>
<td>0x9811 0007</td>
<td>ERR_TARGETMACHINENOTFOUND</td>
<td>Target computer not found – AMS route was not found.</td>
</tr>
<tr>
<td>0x8</td>
<td>8</td>
<td>0x9811 0008</td>
<td>ERR_UNKNOWNCMDID</td>
<td>Unknown command ID.</td>
</tr>
<tr>
<td>0x9</td>
<td>9</td>
<td>0x9811 0009</td>
<td>ERR_BADTASKID</td>
<td>Invalid task ID.</td>
</tr>
<tr>
<td>0xA</td>
<td>10</td>
<td>0x9811 000A</td>
<td>ERR_NOIO</td>
<td>No IO.</td>
</tr>
<tr>
<td>0xB</td>
<td>11</td>
<td>0x9811 000B</td>
<td>ERR_UNKNOWNAMSCMD</td>
<td>Unknown AMS command.</td>
</tr>
<tr>
<td>0xC</td>
<td>12</td>
<td>0x9811 000C</td>
<td>ERR_WIN32ERROR</td>
<td>Win32 error.</td>
</tr>
<tr>
<td>0xD</td>
<td>13</td>
<td>0x9811 000D</td>
<td>ERR_PORTNOTCONNECTED</td>
<td>Port not connected.</td>
</tr>
<tr>
<td>0xE</td>
<td>14</td>
<td>0x9811 000E</td>
<td>ERR_INVALIDAMSLength</td>
<td>Invalid AMS length.</td>
</tr>
<tr>
<td>0xF</td>
<td>15</td>
<td>0x9811 000F</td>
<td>ERR_INVALIDAMSNETID</td>
<td>Invalid AMS Net ID.</td>
</tr>
<tr>
<td>0x10</td>
<td>16</td>
<td>0x9811 0010</td>
<td>ERR_LOWINSTLEVEL</td>
<td>Installation level is too low – TwinCAT 2 license error.</td>
</tr>
<tr>
<td>0x11</td>
<td>17</td>
<td>0x9811 0011</td>
<td>ERR_NODEBUGINTAVAILABLE</td>
<td>No debugging available.</td>
</tr>
<tr>
<td>0x12</td>
<td>18</td>
<td>0x9811 0012</td>
<td>ERR_PORTDISABLED</td>
<td>Port disabled – TwinCAT system service not started.</td>
</tr>
<tr>
<td>0x13</td>
<td>19</td>
<td>0x9811 0013</td>
<td>ERR_PORTALREADYCONNECTED</td>
<td>Port already connected.</td>
</tr>
<tr>
<td>0x14</td>
<td>20</td>
<td>0x9811 0014</td>
<td>ERR_AMSSYNC_W32ERROR</td>
<td>AMS Sync Win32 error.</td>
</tr>
<tr>
<td>0x15</td>
<td>21</td>
<td>0x9811 0015</td>
<td>ERR_AMSSYNC_TIMEOUT</td>
<td>AMS Sync Timeout.</td>
</tr>
<tr>
<td>0x16</td>
<td>22</td>
<td>0x9811 0016</td>
<td>ERR_AMSSYNC_AMSSERROR</td>
<td>AMS Sync error.</td>
</tr>
<tr>
<td>0x17</td>
<td>23</td>
<td>0x9811 0017</td>
<td>ERR_AMSSYNC_NOINDEXINMAP</td>
<td>No index map for AMS Sync available.</td>
</tr>
<tr>
<td>0x18</td>
<td>24</td>
<td>0x9811 0018</td>
<td>ERR_INVALIDAMSPORT</td>
<td>Invalid AMS port.</td>
</tr>
<tr>
<td>0x19</td>
<td>25</td>
<td>0x9811 0019</td>
<td>ERR_NOMEMORY</td>
<td>No memory.</td>
</tr>
<tr>
<td>0x1A</td>
<td>26</td>
<td>0x9811 001A</td>
<td>ERR_TCPSEND</td>
<td>TCP send error.</td>
</tr>
<tr>
<td>0x1B</td>
<td>27</td>
<td>0x9811 001B</td>
<td>ERR_HOSTUNREACHABLE</td>
<td>Host unreachable.</td>
</tr>
<tr>
<td>0x1C</td>
<td>28</td>
<td>0x9811 001C</td>
<td>ERR_INVALIDAMSFRAAGMENT</td>
<td>Invalid AMS fragment.</td>
</tr>
<tr>
<td>0x1D</td>
<td>29</td>
<td>0x9811 001D</td>
<td>ERR_TLSSEND</td>
<td>TLS send error – secure ADS connection failed.</td>
</tr>
<tr>
<td>0x1E</td>
<td>30</td>
<td>0x9811 001E</td>
<td>ERR_ACCESSDENIED</td>
<td>Access denied – secure ADS access denied.</td>
</tr>
</tbody>
</table>

Router error codes
<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
<th>HRESULT</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x500</td>
<td>1280</td>
<td>0x9811 0500</td>
<td>ROUTERERR_NOLOCKEDMEMORY</td>
<td>Locked memory cannot be allocated.</td>
</tr>
<tr>
<td>0x501</td>
<td>1281</td>
<td>0x9811 0501</td>
<td>ROUTERERR_RESIZEMEMORY</td>
<td>The router memory size could not be changed.</td>
</tr>
<tr>
<td>0x502</td>
<td>1282</td>
<td>0x9811 0502</td>
<td>ROUTERERR_MAILBOXFULL</td>
<td>The mailbox has reached the maximum number of possible messages.</td>
</tr>
<tr>
<td>0x503</td>
<td>1283</td>
<td>0x9811 0503</td>
<td>ROUTERERR_DEBUGBOXFULL</td>
<td>The Debug mailbox has reached the maximum number of possible messages.</td>
</tr>
<tr>
<td>0x504</td>
<td>1284</td>
<td>0x9811 0504</td>
<td>ROUTERERR_UNKNOWNPORTTYPE</td>
<td>The port type is unknown.</td>
</tr>
<tr>
<td>0x505</td>
<td>1285</td>
<td>0x9811 0505</td>
<td>ROUTERERR_NOTINITIALIZED</td>
<td>The router is not initialized.</td>
</tr>
<tr>
<td>0x506</td>
<td>1286</td>
<td>0x9811 0506</td>
<td>ROUTERERR_PORTALREADYINUSE</td>
<td>The port number is already assigned.</td>
</tr>
<tr>
<td>0x507</td>
<td>1287</td>
<td>0x9811 0507</td>
<td>ROUTERERR_NOTREGISTERED</td>
<td>The port is not registered.</td>
</tr>
<tr>
<td>0x508</td>
<td>1288</td>
<td>0x9811 0508</td>
<td>ROUTERERR_NOMOREQUEUES</td>
<td>The maximum number of ports has been reached.</td>
</tr>
<tr>
<td>0x509</td>
<td>1289</td>
<td>0x9811 0509</td>
<td>ROUTERERR_INVALIDPORT</td>
<td>The port is invalid.</td>
</tr>
<tr>
<td>0x50A</td>
<td>1290</td>
<td>0x9811 050A</td>
<td>ROUTERERR_NOTACTIVATED</td>
<td>The router is not active.</td>
</tr>
<tr>
<td>0x50B</td>
<td>1291</td>
<td>0x9811 050B</td>
<td>ROUTERERR_FRAGMENTBOXFULL</td>
<td>The mailbox has reached the maximum number for fragmented messages.</td>
</tr>
<tr>
<td>0x50C</td>
<td>1292</td>
<td>0x9811 050C</td>
<td>ROUTERERR_FRAGMENTTIMEOUT</td>
<td>A fragment timeout has occurred.</td>
</tr>
<tr>
<td>0x50D</td>
<td>1293</td>
<td>0x9811 050D</td>
<td>ROUTERERR_TOBEREMOVED</td>
<td>The port is removed.</td>
</tr>
</tbody>
</table>

**General ADS error codes**
<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
<th>HRESULT</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x700</td>
<td>1792</td>
<td>0x9811 0700</td>
<td>ADSERR_DEVICE_ERROR</td>
<td>General device error.</td>
</tr>
<tr>
<td>0x701</td>
<td>1793</td>
<td>0x9811 0701</td>
<td>ADSERR_DEVICE_SRVNOTSUPP</td>
<td>Service is not supported by the server.</td>
</tr>
<tr>
<td>0x702</td>
<td>1794</td>
<td>0x9811 0702</td>
<td>ADSERRDEVICEINVALIDGRP</td>
<td>Invalid index group.</td>
</tr>
<tr>
<td>0x703</td>
<td>1795</td>
<td>0x9811 0703</td>
<td>ADSERR_DEVICE_INVALIDOFFSET</td>
<td>Invalid index offset.</td>
</tr>
<tr>
<td>0x704</td>
<td>1796</td>
<td>0x9811 0704</td>
<td>ADSERR_DEVICE_INVALIDACCESS</td>
<td>Reading or writing not permitted.</td>
</tr>
<tr>
<td>0x705</td>
<td>1797</td>
<td>0x9811 0705</td>
<td>ADSERR_DEVICE_INVALIDSIZE</td>
<td>Parameter size not correct.</td>
</tr>
<tr>
<td>0x706</td>
<td>1798</td>
<td>0x9811 0706</td>
<td>ADSERR_DEVICE_INVALIDDATA</td>
<td>Invalid data values.</td>
</tr>
<tr>
<td>0x707</td>
<td>1799</td>
<td>0x9811 0707</td>
<td>ADSERR_DEVICE_NOTREADY</td>
<td>Device is not ready to operate.</td>
</tr>
<tr>
<td>0x708</td>
<td>1800</td>
<td>0x9811 0708</td>
<td>ADSERR_DEVICE_BUSY</td>
<td>Device is busy.</td>
</tr>
<tr>
<td>0x709</td>
<td>1801</td>
<td>0x9811 0709</td>
<td>ADSERR_DEVICE_INVALIDCONTEXT</td>
<td>Invalid operating system context. This can result from use of ADS function blocks in different tasks. It may be possible to resolve this through multitasking synchronization in the PLC.</td>
</tr>
<tr>
<td>0x70A</td>
<td>1802</td>
<td>0x9811 070A</td>
<td>ADSERR_DEVICE_NOMEMORY</td>
<td>Insufficient memory.</td>
</tr>
<tr>
<td>0x70B</td>
<td>1803</td>
<td>0x9811 070B</td>
<td>ADSERR_DEVICE_INVALIDPARAM</td>
<td>Invalid parameter values.</td>
</tr>
<tr>
<td>0x70C</td>
<td>1804</td>
<td>0x9811 070C</td>
<td>ADSERR_DEVICE_NOTFOUND</td>
<td>Not found (files,...).</td>
</tr>
<tr>
<td>0x70D</td>
<td>1805</td>
<td>0x9811 070D</td>
<td>ADSERR_DEVICE_SYNTAX</td>
<td>Syntax error in file or command.</td>
</tr>
<tr>
<td>0x70E</td>
<td>1806</td>
<td>0x9811 070E</td>
<td>ADSERR_DEVICE_INCOMPATIBLE</td>
<td>Objects do not match.</td>
</tr>
<tr>
<td>0x70F</td>
<td>1807</td>
<td>0x9811 070F</td>
<td>ADSERR_DEVICE_EXISTS</td>
<td>Object already exists.</td>
</tr>
<tr>
<td>0x710</td>
<td>1808</td>
<td>0x9811 0710</td>
<td>ADSERR_DEVICE_SYMBOLNOTFOUND</td>
<td>Symbol not found.</td>
</tr>
<tr>
<td>0x711</td>
<td>1809</td>
<td>0x9811 0711</td>
<td>ADSERR_DEVICE_SYMBOLVERSIONINVALID</td>
<td>Invalid symbol version. This can occur due to an online change. Create a new handle.</td>
</tr>
<tr>
<td>0x712</td>
<td>1810</td>
<td>0x9811 0712</td>
<td>ADSERR_DEVICE_INVALIDSTATE</td>
<td>Device (server) is in invalid state.</td>
</tr>
<tr>
<td>0x713</td>
<td>1811</td>
<td>0x9811 0713</td>
<td>ADSERR_DEVICE_TRANSMODENOTSUPP</td>
<td>AdsTransMode not supported.</td>
</tr>
<tr>
<td>0x714</td>
<td>1812</td>
<td>0x9811 0714</td>
<td>ADSERR_DEVICE_NOTIFYHWNDINVALID</td>
<td>Notification handle is invalid.</td>
</tr>
<tr>
<td>0x715</td>
<td>1813</td>
<td>0x9811 0715</td>
<td>ADSERR_DEVICE_CLIENTUNKNOWN</td>
<td>Notification client not registered.</td>
</tr>
<tr>
<td>0x716</td>
<td>1814</td>
<td>0x9811 0716</td>
<td>ADSERR_DEVICE_NOMEMOREHDL</td>
<td>No further notification handle available.</td>
</tr>
<tr>
<td>0x717</td>
<td>1815</td>
<td>0x9811 0717</td>
<td>ADSERR_DEVICE_INVALIDWATCHSIZE</td>
<td>Notification size too large.</td>
</tr>
<tr>
<td>0x718</td>
<td>1816</td>
<td>0x9811 0718</td>
<td>ADSERR_DEVICE_NOTINIT</td>
<td>Device not initialized.</td>
</tr>
<tr>
<td>0x719</td>
<td>1817</td>
<td>0x9811 0719</td>
<td>ADSERR_DEVICE_TIMEOUT</td>
<td>Device has a timeout.</td>
</tr>
<tr>
<td>0x71A</td>
<td>1818</td>
<td>0x9811 071A</td>
<td>ADSERR_DEVICE_NOINTERFACE</td>
<td>Interface query failed.</td>
</tr>
<tr>
<td>0x71B</td>
<td>1819</td>
<td>0x9811 071B</td>
<td>ADSERR_DEVICE_INVALIDINTERFACE</td>
<td>Wrong interface requested.</td>
</tr>
<tr>
<td>0x71C</td>
<td>1820</td>
<td>0x9811 071C</td>
<td>ADSERR_DEVICE_INVALIDCLSID</td>
<td>Class ID is invalid.</td>
</tr>
<tr>
<td>0x71D</td>
<td>1821</td>
<td>0x9811 071D</td>
<td>ADSERR_DEVICE_INVALIDOBJID</td>
<td>Object ID is invalid.</td>
</tr>
<tr>
<td>0x71E</td>
<td>1822</td>
<td>0x9811 071E</td>
<td>ADSERR_DEVICE_PENDING</td>
<td>Request pending.</td>
</tr>
<tr>
<td>0x71F</td>
<td>1823</td>
<td>0x9811 071F</td>
<td>ADSERR_DEVICE_ABORTED</td>
<td>Request is aborted.</td>
</tr>
<tr>
<td>0x720</td>
<td>1824</td>
<td>0x9811 0720</td>
<td>ADSERR_DEVICE_WARNING</td>
<td>Signal warning.</td>
</tr>
<tr>
<td>0x721</td>
<td>1825</td>
<td>0x9811 0721</td>
<td>ADSERR_DEVICE_INVALIDARRAYIDX</td>
<td>Invalid array index.</td>
</tr>
<tr>
<td>0x722</td>
<td>1826</td>
<td>0x9811 0722</td>
<td>ADSERR_DEVICE_SYMBOLNOTACTIVE</td>
<td>Symbol not active.</td>
</tr>
<tr>
<td>0x723</td>
<td>1827</td>
<td>0x9811 0723</td>
<td>ADSERR_DEVICE_ACCESSDENIED</td>
<td>Access denied.</td>
</tr>
<tr>
<td>0x724</td>
<td>1828</td>
<td>0x9811 0724</td>
<td>ADSERR_DEVICE_LICENSENOTFOUND</td>
<td>Missing license.</td>
</tr>
<tr>
<td>0x725</td>
<td>1829</td>
<td>0x9811 0725</td>
<td>ADSERR_DEVICE_LICENSEEXPIRED</td>
<td>License expired.</td>
</tr>
<tr>
<td>0x726</td>
<td>1830</td>
<td>0x9811 0726</td>
<td>ADSERR_DEVICE_LICENSEEXCEEDED</td>
<td>License exceeded.</td>
</tr>
<tr>
<td>0x727</td>
<td>1831</td>
<td>0x9811 0727</td>
<td>ADSERR_DEVICE_LICENSEINVALID</td>
<td>Invalid license.</td>
</tr>
<tr>
<td>0x728</td>
<td>1832</td>
<td>0x9811 0728</td>
<td>ADSERR_DEVICE_LICENSESYSTEMID</td>
<td>License problem: System ID is invalid.</td>
</tr>
<tr>
<td>0x729</td>
<td>1833</td>
<td>0x9811 0729</td>
<td>ADSERR_DEVICE_LICENSENOTETIMELIMIT</td>
<td>License not limited in time.</td>
</tr>
<tr>
<td>0x72A</td>
<td>1834</td>
<td>0x9811 072A</td>
<td>ADSERR_DEVICE_LICENSEFUTUREISSUE</td>
<td>License problem: Time in the future.</td>
</tr>
<tr>
<td>0x72B</td>
<td>1835</td>
<td>0x9811 072B</td>
<td>ADSERR_DEVICE_LICENSETIMEOUTLONG</td>
<td>License period too long.</td>
</tr>
<tr>
<td>0x72C</td>
<td>1836</td>
<td>0x9811 072C</td>
<td>ADSERR_DEVICE_EXCEPTION</td>
<td>Exception at system startup.</td>
</tr>
<tr>
<td>0x72D</td>
<td>1837</td>
<td>0x9811 072D</td>
<td>ADSERR_DEVICE_LICENSEDEPRECATED</td>
<td>License file read twice.</td>
</tr>
<tr>
<td>0x72E</td>
<td>1838</td>
<td>0x9811 072E</td>
<td>ADSERR_DEVICE_SIGNATUREINVALID</td>
<td>Invalid signature.</td>
</tr>
<tr>
<td>0x72F</td>
<td>1839</td>
<td>0x9811 072F</td>
<td>ADSERR_DEVICE_CERTIFICATEINVALID</td>
<td>Invalid certificate.</td>
</tr>
<tr>
<td>0x730</td>
<td>1840</td>
<td>0x9811 0730</td>
<td>ADSERR_DEVICE_LICENSEOEMNOTFOUND</td>
<td>Public key not known from OEM.</td>
</tr>
<tr>
<td>0x731</td>
<td>1841</td>
<td>0x9811 0731</td>
<td>ADSERR_DEVICE_LICENSERESTRICTED</td>
<td>License not valid for this system ID.</td>
</tr>
<tr>
<td>0x732</td>
<td>1842</td>
<td>0x9811 0732</td>
<td>ADSERR_DEVICE_LICENSEDEMOREDENIED</td>
<td>Demo license prohibited.</td>
</tr>
<tr>
<td>0x733</td>
<td>1843</td>
<td>0x9811 0733</td>
<td>ADSERR_DEVICE_INVALIDFCID</td>
<td>Invalid function ID.</td>
</tr>
<tr>
<td>0x734</td>
<td>1844</td>
<td>0x9811 0734</td>
<td>ADSERR_DEVICE_OUTFORANGE</td>
<td>Outside the valid range.</td>
</tr>
<tr>
<td>0x735</td>
<td>1845</td>
<td>0x9811 0735</td>
<td>ADSERR_DEVICE_INVALIDALIGNMENT</td>
<td>Invalid alignment.</td>
</tr>
</tbody>
</table>
### TCP Winsock error codes

<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x274C</td>
<td>10060</td>
<td>WSAETIMEDOUT</td>
<td>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.</td>
</tr>
<tr>
<td>0x274D</td>
<td>10061</td>
<td>WSAECONNREFUSED</td>
<td>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.</td>
</tr>
<tr>
<td>0x2751</td>
<td>10065</td>
<td>WSAEHOSTUNREACH</td>
<td>No route to host - a socket operation referred to an unavailable host.</td>
</tr>
</tbody>
</table>

### RTime error codes

<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
<th>HRESULT</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1009</td>
<td>4100</td>
<td>0x8111 1004</td>
<td>RTERR_PRIOREXISTS</td>
<td>The request task priority is already assigned.</td>
</tr>
<tr>
<td>0x100F</td>
<td>4111</td>
<td>0x8111 100F</td>
<td>RTERR_EXTIROINSTALLFAILED</td>
<td>Application of the external synchronization interrupt has failed.</td>
</tr>
<tr>
<td>0x1012</td>
<td>4112</td>
<td>0x8111 1010</td>
<td>RTERR_IRQLNOTLESSOREQUAL</td>
<td>Call of a service function in the wrong context</td>
</tr>
<tr>
<td>0x1017</td>
<td>4117</td>
<td>0x8111 1017</td>
<td>RTERR_VMXNOTSUPPORTED</td>
<td>Intel VT-x extension is not supported.</td>
</tr>
<tr>
<td>0x1018</td>
<td>4120</td>
<td>0x8111 1018</td>
<td>RTERR_VMXDISABLED</td>
<td>Intel VT-x extension is not enabled in the BIOS.</td>
</tr>
<tr>
<td>0x1019</td>
<td>4121</td>
<td>0x8111 1019</td>
<td>RTERR_VMXCONTROLSMISSING</td>
<td>Missing function in Intel VT-x extension.</td>
</tr>
<tr>
<td>0x101A</td>
<td>4122</td>
<td>0x8111 101A</td>
<td>RTERR_VMXENABLEFAILS</td>
<td>Activation of Intel VT-x fails.</td>
</tr>
<tr>
<td>0x1006</td>
<td>4102</td>
<td>0x8111 1006</td>
<td>RTERR_NOMORETCB</td>
<td>No free TCB (Task Control Block) available. The maximum number of TCBs is 64.</td>
</tr>
<tr>
<td>0x1007</td>
<td>4103</td>
<td>0x8111 1007</td>
<td>RTERR_NOMOREQUEUES</td>
<td>No free space available. The maximum number of semaphores is 64.</td>
</tr>
<tr>
<td>0x100D</td>
<td>4109</td>
<td>0x8111 100D</td>
<td>RTERR_EXTERNALREADYDEF</td>
<td>An external synchronization interrupt is already applied.</td>
</tr>
<tr>
<td>0x100E</td>
<td>4110</td>
<td>0x8111 100E</td>
<td>RTERR_EXTERNALNOTDEF</td>
<td>No external sync interrupt applied.</td>
</tr>
<tr>
<td>0x1010</td>
<td>4112</td>
<td>0x8111 1010</td>
<td>RTERR_EXTERNALNOTDISABLE</td>
<td>Call of a service function in the wrong context</td>
</tr>
<tr>
<td>0x1017</td>
<td>4117</td>
<td>0x8111 1017</td>
<td>RTERR_VMXNOTSUPPORTED</td>
<td>Intel VT-x extension is not supported.</td>
</tr>
<tr>
<td>0x1018</td>
<td>4120</td>
<td>0x8111 1018</td>
<td>RTERR_VMXDISABLED</td>
<td>Intel VT-x extension is not enabled in the BIOS.</td>
</tr>
<tr>
<td>0x1019</td>
<td>4121</td>
<td>0x8111 1019</td>
<td>RTERR_VMXCONTROLSMISSING</td>
<td>Missing function in Intel VT-x extension.</td>
</tr>
<tr>
<td>0x101A</td>
<td>4122</td>
<td>0x8111 101A</td>
<td>RTERR_VMXENABLEFAILS</td>
<td>Activation of Intel VT-x fails.</td>
</tr>
<tr>
<td>0x1005</td>
<td>4101</td>
<td>0x8111 1005</td>
<td>RTERR_NOMORETCB</td>
<td>No free TCB (Task Control Block) available. The maximum number of TCBs is 64.</td>
</tr>
<tr>
<td>0x1006</td>
<td>4102</td>
<td>0x8111 1006</td>
<td>RTERR_NOMORETCB</td>
<td>No free TCB (Task Control Block) available. The maximum number of TCBs is 64.</td>
</tr>
<tr>
<td>0x1007</td>
<td>4103</td>
<td>0x8111 1007</td>
<td>RTERR_NOMOREQUEUES</td>
<td>No free space available. The maximum number of semaphores is 64.</td>
</tr>
<tr>
<td>0x100D</td>
<td>4109</td>
<td>0x8111 100D</td>
<td>RTERR_EXTERNALREADYDEF</td>
<td>An external synchronization interrupt is already applied.</td>
</tr>
<tr>
<td>0x100E</td>
<td>4110</td>
<td>0x8111 100E</td>
<td>RTERR_EXTERNALNOTDEF</td>
<td>No external sync interrupt applied.</td>
</tr>
<tr>
<td>0x100F</td>
<td>4111</td>
<td>0x8111 100F</td>
<td>RTERR_EXTERNALNOTDISABLE</td>
<td>Call of a service function in the wrong context</td>
</tr>
<tr>
<td>0x1010</td>
<td>4112</td>
<td>0x8111 1010</td>
<td>RTERR_EXTERNALNOTDISABLE</td>
<td>Call of a service function in the wrong context</td>
</tr>
<tr>
<td>0x1017</td>
<td>4117</td>
<td>0x8111 1017</td>
<td>RTERR_VMXNOTSUPPORTED</td>
<td>Intel VT-x extension is not supported.</td>
</tr>
<tr>
<td>0x1018</td>
<td>4120</td>
<td>0x8111 1018</td>
<td>RTERR_VMXDISABLED</td>
<td>Intel VT-x extension is not enabled in the BIOS.</td>
</tr>
<tr>
<td>0x1019</td>
<td>4121</td>
<td>0x8111 1019</td>
<td>RTERR_VMXCONTROLSMISSING</td>
<td>Missing function in Intel VT-x extension.</td>
</tr>
<tr>
<td>0x101A</td>
<td>4122</td>
<td>0x8111 101A</td>
<td>RTERR_VMXENABLEFAILS</td>
<td>Activation of Intel VT-x fails.</td>
</tr>
</tbody>
</table>

### Win32 error codes

<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x751</td>
<td>10065</td>
<td>WSAEHOSTUNREACH</td>
<td>No route to host - a socket operation referred to an unavailable host.</td>
</tr>
</tbody>
</table>

### More Winsock error codes: Win32 error codes

- 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.
- 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.
- WSADERELEASEFAILS: Activation of Intel VT-x fails.

### Version: 1.8
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_INVALID:=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_EAT:=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_SET_UP:=24,
    HTTP_REQ_ERR_TLS_BANDSHARE_FAILED:=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_INVALID:=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_EXT_NAME:=1106,
    HTTP_REQ_ERR_PARSE_CHUNK_EXT_VAL:=1107,
    HTTP_REQ_ERR_PARSE_CHUNK_DATA:=1108,
    HTTP_REQ_ERR_PARSE_CHUNK_TRAILER:=1110
) DINT;
END_TYPE
7.2.2  ETcIoHttpHttpRequestType

TYPE ETcIoHttpHttpRequestType:
{
    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_READ:=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.
<table>
<thead>
<tr>
<th>Cipher Suite</th>
</tr>
</thead>
<tbody>
<tr>
<td>DHE-RSA-AES256-SHA256</td>
</tr>
<tr>
<td>ECDHE-ECDSA-AES256-SHA</td>
</tr>
<tr>
<td>ECDHE-RSA-AES256-SHA</td>
</tr>
<tr>
<td>DHE-RSA-AES256-SHA</td>
</tr>
<tr>
<td>ECDHE-ECDSA-AES128-GCM-SHA256</td>
</tr>
<tr>
<td>ECDHE-RSA-AES128-GCM-SHA256</td>
</tr>
<tr>
<td>DHE-RSA-AES128-GCM-SHA256</td>
</tr>
<tr>
<td>ECDHE-ECDSA-AES128-SHA256</td>
</tr>
<tr>
<td>DHE-RSA-AES128-SHA256</td>
</tr>
<tr>
<td>ECDHE-ECDSA-AES128-SHA</td>
</tr>
<tr>
<td>DHE-RSA-AES128-SHA</td>
</tr>
<tr>
<td>ECDHE-ECDSA-DES-CBC3-SHA</td>
</tr>
<tr>
<td>ECDHE-RSA-DES-CBC3-SHA</td>
</tr>
<tr>
<td>EDH-RSA-DES-CBC3-SHA</td>
</tr>
<tr>
<td>AES256-SHA256</td>
</tr>
<tr>
<td>AES256-SHA</td>
</tr>
<tr>
<td>AES128-GCM-SHA256</td>
</tr>
<tr>
<td>AES128-SHA256</td>
</tr>
<tr>
<td>AES128-SHA</td>
</tr>
<tr>
<td>DES-CBC3-SHA</td>
</tr>
<tr>
<td>PSK-AES256-CBC-SHA</td>
</tr>
<tr>
<td>PSK-AES128-GCM-SHA256</td>
</tr>
<tr>
<td>PSK-AES128-CBC-SHA256</td>
</tr>
<tr>
<td>PSK-AES128-CBC-SHA</td>
</tr>
<tr>
<td>PSK-3DES-EDE-CBC-SHA</td>
</tr>
</tbody>
</table>