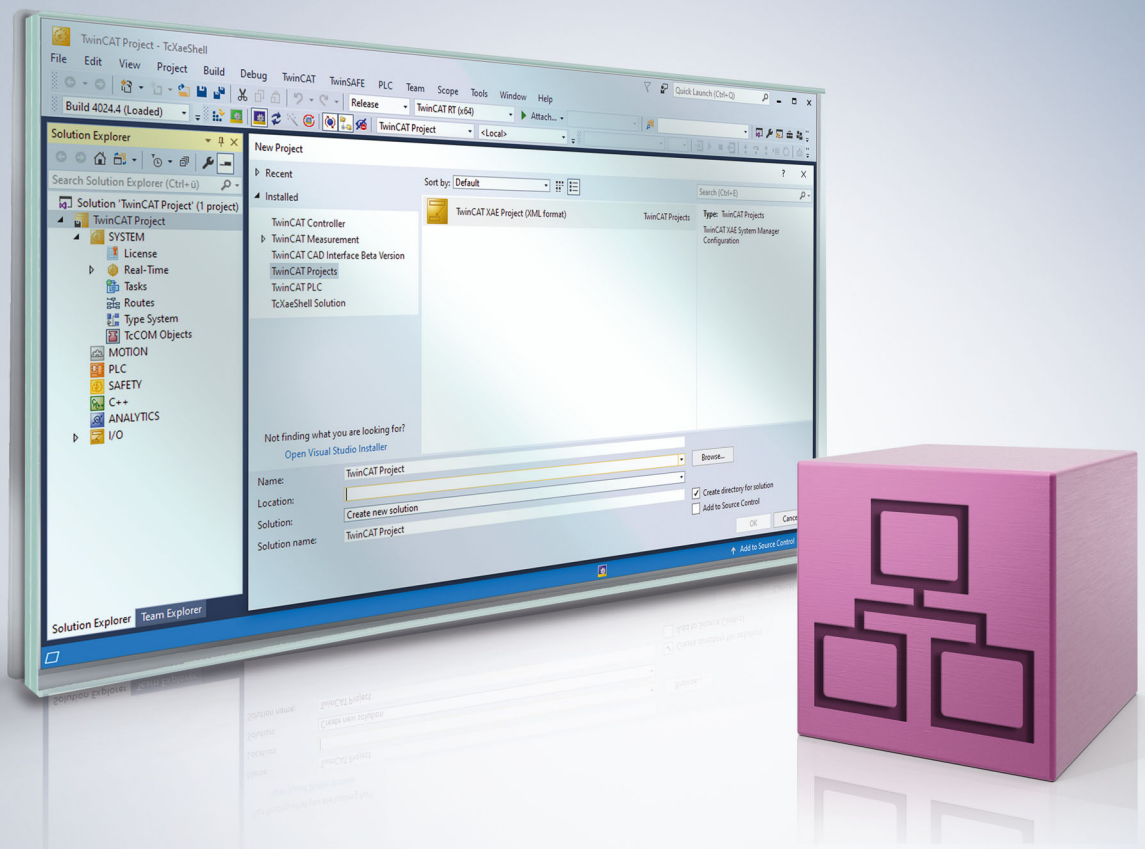
BECKHOFF New Automation Technology

Handbuch | DE

TF6760

TwinCAT 3 | IoT HTTPS/REST



19.09.2022 | Version: 1.9

Inhaltsverzeichnis

1 Vorwort

1.1 Hinweise zur Dokumentation

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs- und Automatisierungstechnik, das mit den geltenden nationalen Normen vertraut ist.
Zur Installation und Inbetriebnahme der Komponenten ist die Beachtung der Dokumentation und der nachfolgenden Hinweise und Erklärungen unbedingt notwendig.
Das Fachpersonal ist verpflichtet, für jede Installation und Inbetriebnahme die zu dem betreffenden Zeitpunkt veröffentliche Dokumentation zu verwenden.

Das Fachpersonal hat sicherzustellen, dass die Anwendung bzw. der Einsatz der beschriebenen Produkte alle Sicherheitsanforderungen, einschließlich sämtlicher anwendbaren Gesetze, Vorschriften, Bestimmungen und Normen erfüllt.

Disclaimer

Diese Dokumentation wurde sorgfältig erstellt. Die beschriebenen Produkte werden jedoch ständig weiter entwickelt.
Wir behalten uns das Recht vor, die Dokumentation jederzeit und ohne Ankündigung zu überarbeiten und zu ändern.
Aus den Angaben, Abbildungen und Beschreibungen in dieser Dokumentation können keine Ansprüche auf Änderung bereits gelieferter Produkte geltend gemacht werden.

Marken

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® und XPlanar® sind eingetragene und lizenzierte Marken der Beckhoff Automation GmbH.
Die Verwendung anderer in dieser Dokumentation enthaltenen Marken oder Kennzeichen durch Dritte kann zu einer Verletzung von Rechten der Inhaber der entsprechenden Bezeichnungen führen.

Patente

Die EtherCAT-Technologie ist patentrechtlich geschützt, insbesondere durch folgende Anmeldungen und Patente:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
mit den entsprechenden Anmeldungen und Eintragungen in verschiedenen anderen Ländern.

EtherCAT® ist eine eingetragene Marke und patentierte Technologie lizenziert durch die Beckhoff Automation GmbH, Deutschland

Copyright

1.2 Sicherheitshinweise

Sicherheitsbestimmungen

Beachten Sie die folgenden Sicherheitshinweise und Erklärungen!
Produktspezifische Sicherheitshinweise finden Sie auf den folgenden Seiten oder in den Bereichen Montage, Verdrahtung, Inbetriebnahme usw.

Haftungsausschluss

Die gesamten Komponenten werden je nach Anwendungsbestimmungen in bestimmten Hard- und Software-Konfigurationen ausgeliefert. Änderungen der Hard- oder Software-Konfiguration, die über die dokumentierten Möglichkeiten hinausgehen, sind unzulässig und bewirken den Haftungsausschluss der Beckhoff Automation GmbH & Co. KG.

Qualifikation des Personals

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs-, Automatisierungs- und Antriebstechnik, das mit den geltenden Normen vertraut ist.

Erklärung der Symbole

In der vorliegenden Dokumentation werden die folgenden Symbole mit einem nebenstehenden Sicherheitshinweis oder Hinweistext verwendet. Die Sicherheitshinweise sind aufmerksam zu lesen und unbedingt zu befolgen!

⚠ GEFAHR

Akute Verletzungsgefahr!

Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, besteht unmittelbare Gefahr für Leben und Gesundheit von Personen!

⚠ WARNUNG

Verletzungsgefahr!

Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, besteht Gefahr für Leben und Gesundheit von Personen!

⚠ VORSICHT

Schädigung von Personen!

Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, können Personen geschädigt werden!

HINWEIS

Schädigung von Umwelt oder Geräten

Wenn der Hinweis neben diesem Symbol nicht beachtet wird, können Umwelt oder Geräte geschädigt werden.

Tipp oder Fingerzeig

Dieses Symbol kennzeichnet Informationen, die zum besseren Verständnis beitragen.

1.3 Hinweise zur Informationssicherheit

Die Produkte der Beckhoff Automation GmbH & Co. KG (Beckhoff) sind, sofern sie online zu erreichen sind, mit Security-Funktionen ausgestattet, die den sicheren Betrieb von Anlagen, Systemen, Maschinen und Netzwerken unterstützen. Trotz der Security-Funktionen sind die Erstellung, Implementierung und ständige Aktualisierung eines ganzheitlichen Security-Konzepts für den Betrieb notwendig, um die jeweilige Anlage, das System, die Maschine und die Netzwerke gegen Cyber-Bedrohungen zu schützen. Die von Beckhoff verkauften Produkte bilden dabei nur einen Teil des gesamtheitlichen Security-Konzepts. Der Kunde ist dafür verantwortlich, dass unbefugte Zugriffe durch Dritte auf seine Anlagen, Systeme, Maschinen und Netzwerke verhindert werden. Letztere sollten nur mit dem Unternehmensnetzwerk oder dem Internet verbunden werden, wenn entsprechende Schutzmaßnahmen eingerichtet wurden.

Zusätzlich sollten die Empfehlungen von Beckhoff zu entsprechenden Schutzmaßnahmen beachtet werden. Weiterführende Informationen über Informationssicherheit und Industrial Security finden Sie in unserem https://www.beckhoff.de/secguide.

Die Produkte und Lösungen von Beckhoff werden ständig weiterentwickelt. Dies betrifft auch die Security-Funktionen. Aufgrund der stetigen Weiterentwicklung empfiehlt Beckhoff ausdrücklich, die Produkte ständig auf dem aktuellen Stand zu halten und nach Bereitstellung von Updates diese auf die Produkte aufzuspielen. Die Verwendung veralteter oder nicht mehr unterstützter Produktversionen kann das Risiko von Cyber-Bedrohungen erhöhen.

Um stets über Hinweise zur Informationssicherheit zu Produkten von Beckhoff informiert zu sein, abonnieren Sie den RSS Feed unter https://www.beckhoff.de/secinfo.

2 Übersicht

Die Funktionsbausteine der SPS-Bibliothek Tc3_IoTBase können verwendet werden, um REST-APIs über HTTP/HTTPS-Kommunikation als Client anzusprechen. Hierzu stehen gängige HTTP-Befehle wie GET, PUT oder POST zur Verfügung. Grundsätzlich können Daten von einer REST API angefordert und auch an eine REST-API gesendet werden.

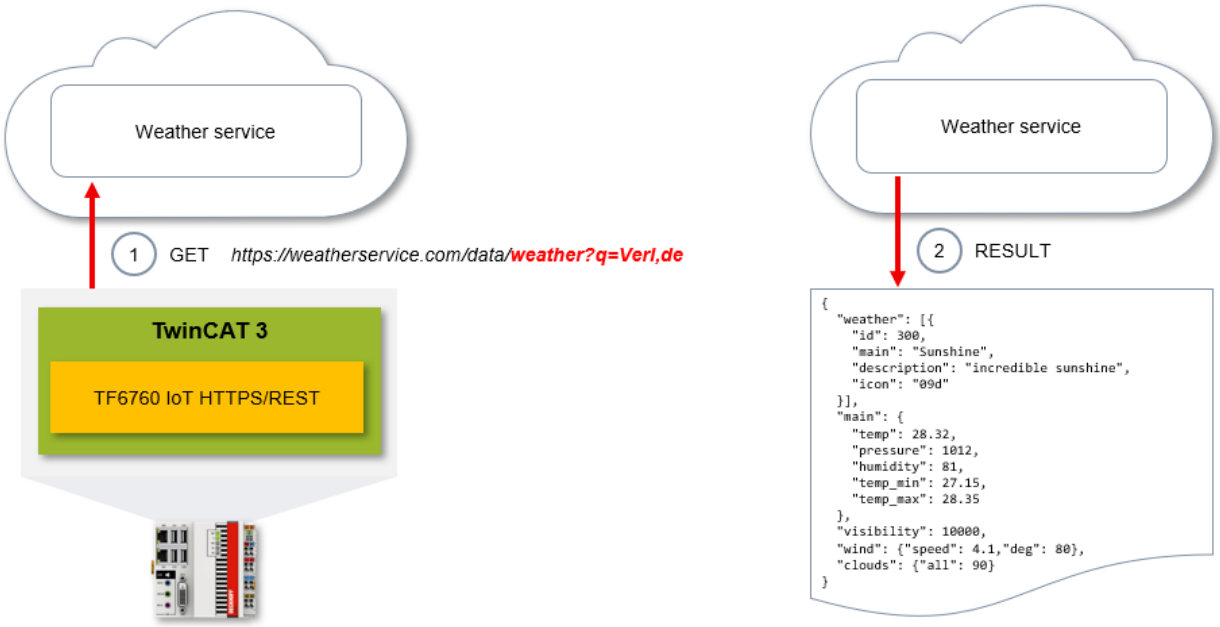


Abb. 1: TF6760

Produktkomponenten

Die Funktion TF6760 IoT HTTPS/REST besteht aus den folgenden Komponenten, die standardmäßig im Lieferumfang von TwinCAT 3 enthalten sind:

- **Treiber:** TcIotDrivers.sys (im Lieferumfang der TwinCAT 3 XAE- und XAR-Setups)
- **SPS-Bibliothek:** Tc3_IotBase (im Lieferumfang des TwinCAT 3 XAE-Setups)

3 Installation

3.1 Systemanforderungen

Technische Daten	Beschreibung
Betriebssystem	Windows 7/10, Windows Embedded Standard 7, Windows CE 7
Zielplattform	PC-Architektur (x86, x64 oder ARM)
TwinCAT-Version	TwinCAT 3.1 Build 4024.7 oder höher
Erforderliches TwinCAT-Setup-Level	TwinCAT 3 XAE, XAR
Erforderliche TwinCAT-Lizenz	TF6760 TC3 IoT HTTPS/REST

3.2 Installation

Für TF6760 IoT HTTPS/REST ist kein separates Setup notwendig. Alle erforderlichen Komponenten werden direkt im Rahmen des TwinCAT-Setups geliefert.

- TwinCAT XAE-Setup: Treiberkomponenten und SPS-Bibliothek (Tc3_IotBase)
- TwinCAT XAR-Setup: Treiberkomponenten

3.3 Lizenzierung

Die TwinCAT 3 Function ist als Vollversion oder als 7-Tage-Testversion freischaltbar. Beide Lizenztypen sind über die TwinCAT-3-Entwicklungsumgebung (XAE) aktivierbar.

Lizenzierung der Vollversion einer TwinCAT 3 Function

Die Beschreibung der Lizenzierung einer Vollversion finden Sie im Beckhoff Information System in der Dokumentation „TwinCAT 3 Lizenzierung“.

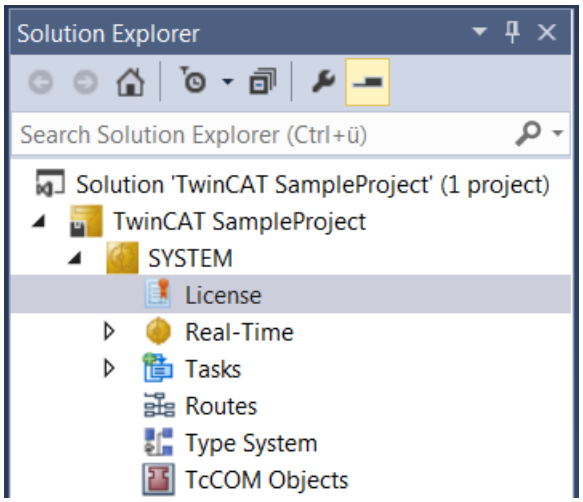
Lizenzierung der 7-Tage-Testversion einer TwinCAT 3 Function

Eine 7-Tage-Testversion kann nicht für einen TwinCAT 3 Lizenzdongle freigeschaltet werden.

1. Starten Sie die TwinCAT-3-Entwicklungsumgebung (XAE).
2. Öffnen Sie ein bestehendes TwinCAT-3-Projekt oder legen Sie ein neues Projekt an.
3. Wenn Sie die Lizenz für ein Remote-Gerät aktivieren wollen, stellen Sie das gewünschte Zielsystem ein. Wählen Sie dazu in der Symbolleiste in der Drop-down-Liste **Choose Target System** das Zielsystem aus.

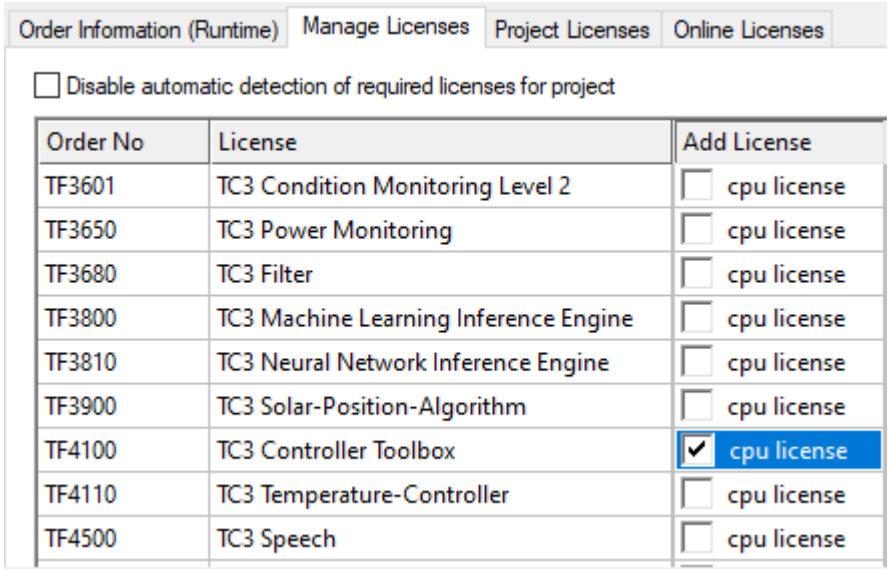
   ⇨ Die Lizenzierungseinstellungen beziehen sich immer auf das eingestellte Zielsystem. Mit der Aktivierung des Projekts auf dem Zielsystem werden automatisch auch die zugehörigen TwinCAT-3-Lizenzen auf dieses System kopiert.

4. Klicken Sie im **Solution Explorer** im Teilbaum **SYSTEM** doppelt auf **License.**



⇨ Der TwinCAT-3-Lizenzmanager öffnet sich.

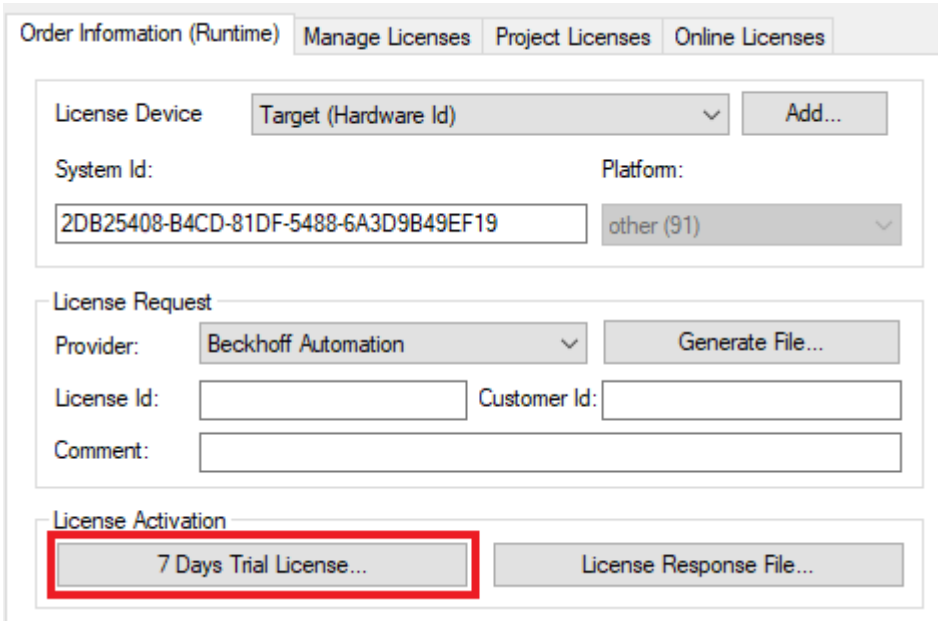
5. Öffnen Sie die Registerkarte **Manage Licenses.** Aktivieren Sie in der Spalte **Add License** das Auswahlkästchen für die Lizenz, die Sie Ihrem Projekt hinzufügen möchten (z. B. „TF4100 TC3 Controller Toolbox“).



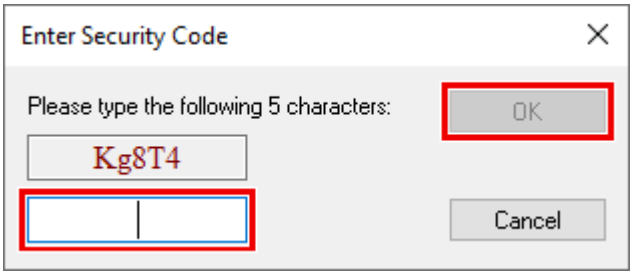
6. Öffnen Sie die Registerkarte **Order Information (Runtime).**

⇨ In der tabellarischen Übersicht der Lizenzen wird die zuvor ausgewählte Lizenz mit dem Status „missing“ angezeigt.

7. Klicken Sie auf **7 Days Trial License...**, um die 7-Tage-Testlizenz zu aktivieren.



⇨ Es öffnet sich ein Dialog, der Sie auffordert, den im Dialog angezeigten Sicherheitscode einzugeben.



8. Geben Sie den Code genauso ein, wie er angezeigt wird, und bestätigen Sie ihn.
9. Bestätigen Sie den nachfolgenden Dialog, der Sie auf die erfolgreiche Aktivierung hinweist.
   ⇨ In der tabellarischen Übersicht der Lizenzen gibt der Lizenzstatus nun das Ablaufdatum der Lizenz an.
10. Starten Sie das TwinCAT-System neu.

⇨ Die 7-Tage-Testversion ist freigeschaltet.

4 Technische Einführung

Das folgende Kapitel enthält eine kurze technische Einführung in das HTTP-Protokoll und seine sichere Erweiterung, das HTTPS-Protokoll. Zudem werden die REST-Architektur und ihr Zusammenhang mit dem HTTP-Protokoll beschrieben.

4.1 HTTP/HTTPS

HTTP (Hypertext Transfer Protocol) ist ein Standardprotokoll, das hauptsächlich für die IP-Kommunikation zwischen Servern und Clients im Internet verwendet wird. Der Client ist in der Regel ein Webbrowser, der eine Website von einem Webserver anfordert. Ein weiterer Anwendungsfall für das HTTP-Protokoll sind REST-Webservices (Representational State Transfer).

Des Weiteren ist HTTP ein zustandsloses Protokoll, jeder Befehl wird unabhängig behandelt. Nachdem ein Server auf eine Client-Anfrage geantwortet hat, wird die Verbindung wieder geschlossen. Ein Client hat die Möglichkeit, dem Server mitzuteilen, dass die Verbindung nach einer Anfrage aufrechterhalten werden soll.

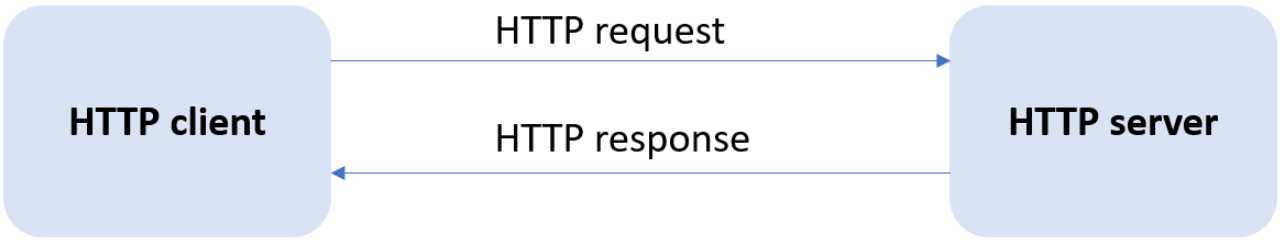


Abb. 2: HTTP-Kommunikation

HTTP-Methoden

Der HTTP-Standard (Version 1.1) definiert verschiedene Methoden, die ein HTTP-Server anfragenden Clients bieten kann. Insbesondere GET, POST und PUT sind einige der am häufigsten verwendeten HTTP-Methoden. Die folgende Tabelle enthält alle HTTP-Methoden, die in dem Standard definiert sind.

HTTP-Methode	Beschreibung
GET	Fordert eine Ressource vom Server an.
POST	Übermittelt Daten an den Server.
PUT	Ersetzt oder erstellt eine Ressource auf dem Server mit der Anfragenutzlast.
CONNECT	Baut einen SSL-Tunnel zu einem Server auf.
DELETE	Löscht die adressierte Ressource auf dem Server.
HEAD	Gleicher Funktionsumfang wie GET ohne Nutzlastdaten.
OPTIONS	Fordert die verfügbaren Kommunikationsoptionen vom Server an.
TRACE	Führt einen Nachrichten-Loopback zu Testzwecken durch.
PATCH	Gleicher Funktionsumfang wie PUT, jedoch verwendet für teilweise Änderungen von Ressourcen.

Ein Benutzer muss wissen, dass nicht jeder Server all diese Methoden implementiert. Gemäß der Spezifikation sind nur GET und HEAD für einen spezifikationsgetreuen Server obligatorisch, alle anderen Befehle sind optional. Zudem besteht die Möglichkeit, dass für den Zugriff auf Ressourcen auf einem Server Anmeldeinformationen erforderlich sind.

HTTP-Statuscodes

Eine HTTP-Antwort ist in einen Header und einen Body unterteilt. Ein wichtiger Teil dieser Antwort ist der HTTP-Statuscode. Der HTTP-Statuscode liefert Informationen über die Anfrage zurück an den Client. Die folgende Tabelle zeigt die definierten Bereiche der HTTP-Statuscodes.

Statuscodebereich	Beschreibung
100-199	Statuscodes, die Informationsmeldungen enthalten.
200-299	Statuscodes, die den Client über eine erfolgreiche Anfrage informieren.
300-399	Statuscodes, die eine Umleitung zeigen und eine Interaktion vom Benutzer verlangen.
400-499	Statuscodes, die Fehlermeldungen zeigen (ausgelöst vom Client).
500-599	Statuscodes, die Fehlermeldungen zeigen (ausgelöst vom Server).

HTTPS

HTTPS (Hypertext Transfer Protocol Secure) ist eine Erweiterung des HTTP-Protokolls, die TLS (Transport Layer Security) implementiert. Daten, die mit dem HTTP-Protokoll gesendet werden, sind vollkommen unverschlüsselt, weshalb es die Sicherheitsanforderungen für das Senden von z. B. Passwörtern oder Kreditkarteninformationen nicht erfüllt. Zusätzlich zur Verschlüsselung der transportierten Daten bietet HTTPS auch Serverauthentifizierung.

4.2 REST-API

Ein RESTful (Representational State Transfer) Webservice ist ein Webdienst, der mit einer Architektur speziell für dezentrale Systeme gestaltet ist. Alle RESTful Webservices zusammen auf einem Server werden als REST-API (Application Programming Interface) bezeichnet.

REST-APIs werden häufig direkt mit dem HTTP-Protokoll in Verbindung gebracht, sind jedoch weder ein Standard noch ein Protokoll. Es gibt – definitionsgemäß – verschiedene Protokolle, die für die Kommunikation mit REST-APIs angewendet werden können, HTTP ist lediglich das gängigste.

Eine REST-API muss sechs verschiedenen Architekturdefinitionen entsprechen, die in der Spezifikation genannt sind:

- **Client-Server-Modell:** Daten und Benutzerschnittstelle müssen definitionsgemäß getrennt sein.
- **Zustandslosigkeit:** Client und Server müssen zustandslos kommunizieren. Jede Client-Anfrage muss alle Informationen enthalten, die der Server für die Verarbeitung der Anfrage benötigt.
- **Caching:** Informationen können als im Cache speicherbar oder nicht im Cache speicherbar eingestuft werden. Ein Client kann Serverantworten für künftige Anfragen im Cache speichern, diese Informationen könnten aber veraltet sein.
- **Einheitliche Schnittstelle:** REST-Schnittstellen müssen aufgrund ihrer einheitlichen Schnittstelle von verschiedenen Endpunkten identisch nutzbar sein.
- **Mehrschichtige Systeme:** Eine REST-API muss ein mehrschichtiges, hierarchisch aufgebautes System sein.
- **Code-On-Demand (optional):** Die Funktionen der Clients müssen mit nachladbaren und ausführbaren Programmteilen erweiterbar sein.

Anforderung von Ressourcen von einer REST-API

Wie bereits erwähnt, ist HTTP(S) nur eine der Möglichkeiten, um eine REST-API zu implementieren, doch es ist die einzige, die für diese Dokumentation von Bedeutung ist. Der folgende Abschnitt enthält eine kurze Beschreibung, wie Ressourcen von einer REST-API angefordert werden, die mit HTTP(S) implementiert ist.

Eine Ressource auf einer HTTPS-REST-API wird mit ihrer URL identifiziert. Es gibt zwei verschiedene Möglichkeiten, wie eine URL von einem Webservice angeboten werden kann. Zum einen kann eine URL nur aus Pfadparametern bestehen und zum anderen könnte sie Abfrageparameter enthalten. Der Unterschied zwischen diesen beiden Möglichkeiten wird am folgenden Beispiel erläutert.

Eine URL ist aus drei Teilen aufgebaut:

1. Domainname
2. Pfadparameter
3. Abfrageparameter

Wenn das Wetter für die Stadt Verl von einem fiktiven Beckhoff-Wetterdienst angefordert werden soll, könnte die URL ohne Abfrageparameter wie folgt aussehen:

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

Mit Abfrageparametern könnte die URL folgendermaßen aussehen:

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

Der erste Teil **https://www.beckhoff-weather.com** ist der Domainname des Webservers, der mit Hilfe eines DNS-Servers in eine IP-Adresse umgewandelt wird. Die Pfadparameter **/forecast/city/verl** geben an, auf welcher Ressource des Servers die Anfrage verarbeitet werden soll. Eine weitere Möglichkeit ist die Verwendung eines Abfrageparameters wie **forecast?city=verl**, um eine spezifische Ressource auf einem Pfad anzufordern.

4.3 Technische Einschränkungen

In diesem Kapitel werden technische Einschränkungen des IoT-Treibers im Bezug auf die HTTP-Client-Funktionalität dargestellt. Bei diesen Einschränkungen handelt es sich in keinem Fall um fehlerhafte Verhaltensweisen, sondern um Verhaltensweisen, die „by-design“ sind und bewusst so festgelegt wurden.

Einschränkung	Beschreibung
Anzahl von gleichzeitigen HTTP Requests von einer HTTP-Client-Instanz	Jede Instanz des Funktionsbausteins FB_IotHttpClient [▶ 23] kann nur einen HTTP-Request gleichzeitig absetzen. Erst wenn die zugehörige HTTP-Response angekommen ist, kann der nächste Request abgesetzt werden. Werden dennoch mehrere Requests vor dem Eintreffen der Response von der gleichen Client-Instanz abgesetzt, werden diese in eine Warteschlange eingeordnet. Übergeordnet über die Response ist der an der HTTP-Client-Instanz konfigurierte Timeout.
Kommunikation von großen Dateien	Bei der Kommunikation von großen Dateien sollte sich der Anwender bewusst sein, dass dies zu einer Belastung der Echtzeit führt, da die große Menge an Daten kopiert werden muss. Außerdem wird Router-Speicher verwendet.

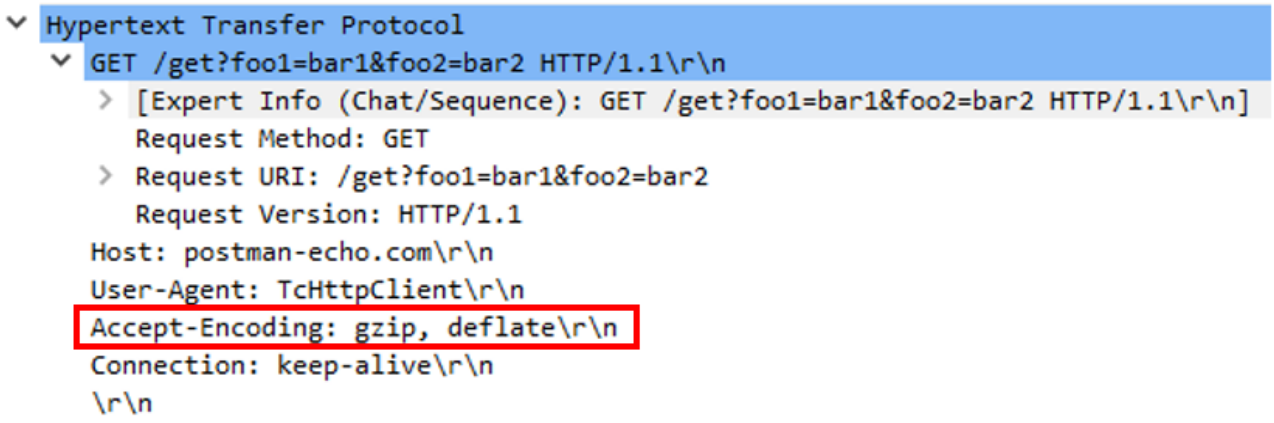
4.4 Kompression

Im Allgemeinen bezeichnet das Wort „Datenkompression“ die Fähigkeit, die Anzahl an Bits zu verringern, die zur Darstellung von Daten notwendig sind. Dazu werden beispielsweise wiederkehrende Zeichenfolgen durch einen Kompressionsalgorithmus mit einer Referenz auf die erste dieser Zeichenfolgen versehen. Eine geeignete Kompression muss ohne Informationsverlust passieren.

Zwei im Bereich der HTTP-Kommunikation sehr verbreitete Kompressionsarten sind gzip und deflate. Beide werden in Sende- und Empfangsrichtung im HTTP-Treiber und der Tc3_IotBase-Bibliothek seit TwinCAT-Version 3.1.4024.11 unterstützt. Die Verfahren gehören zur Gruppe der verlustfreien Kompressionsarten.

In TwinCAT kann die Kompression für einen FB_IotHttpRequest [▶ 26]-Baustein individuell eingestellt werden und wird diesem als Eingangsparameter übergeben. Als Standard wird in Senderichtung keine Kompression verwendet, in Empfangsrichtung kann der Treiber ohne separate Einstellungen beide Arten der Kompression empfangen und korrekt darstellen.

In der folgenden Abbildung wird ein HTTP-Request an die Test-REST API Postman Echo dargestellt (siehe PostmanEcho [▶ 128]). Der TwinCAT HTTP-Client teilt der REST API per Header-Field mit, dass sowohl deflate als auch gzip in Empfangsrichtung unterstützt werden.



4.5 Sicherheit

Bei der Betrachtung der Absicherung einer Datenkommunikation lassen sich zwei Ebenen voneinander unterscheiden. Zum einen die Absicherung des Transportkanals (Transportebene [▶ 15]) und zum anderen die Absicherung auf Applikationsebene (Applikationsebene [▶ 19]). Beide werden im Folgenden beschrieben.

4.5.1 Transportebene

Für die sichere Übertragung von Daten wird im TwinCAT IoT-Treiber der Standard Transport Layer Security (TLS) verwendet. **Das folgende Kapitel beschreibt den TLS-Kommunikationsablauf für die TLS-Version 1.2**. Der TLS-Standard ist eine Kombination aus symmetrischer und asymmetrischer Kryptographie und schützt versendete Daten vor unbefugtem Zugriff und Manipulation durch Dritte. Außerdem wird die Authentifizierung der Kommunikationsteilnehmer zur gegenseitigen Identitätsprüfung von TLS unterstützt.

Inhalt dieses Kapitels

Die folgenden Informationen in diesem Kapitel beziehen sich in allgemeiner Weise auf den TLS-Kommunikationsablauf und haben keinen Bezug zu der Implementierung in TwinCAT. Sie sollen lediglich für ein grundlegendes Verständnis sorgen, um den in den folgenden Unterkapiteln erläuterten Bezug zu der TwinCAT-Implementierung besser nachvollziehen zu können.

Definition Cipher-Suite

Eine Cipher Suite nach Definition der TLS-Version 1.2 ist eine Zusammensetzung von Algorithmen (Schlüsselaustausch, Authentifizierung, Verschlüsselung, MAC) zur Verschlüsselung. Auf diese einigen sich Client und Server während des TLS-Verbindungsaufbaus. Weitere Informationen zu Cipher Suites entnehmen Sie bitte der Fachliteratur.

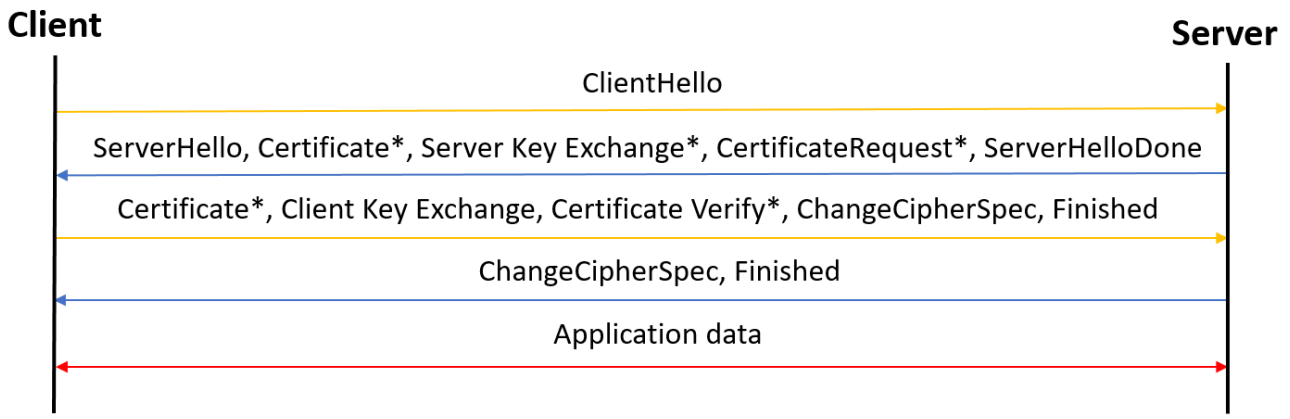
TLS-Kommunikationsablauf

Am Anfang einer Kommunikation mit TLS-Verschlüsselung steht der sogenannte TLS-Handshake zwischen Server und Client. Während des Handshakes wird asymmetrische Kryptographie verwendet, nach erfolgreichem Abschluss des Handshakes kommunizieren Server und Client mit symmetrischer Kryptographie, da diese um ein Vielfaches schneller ist als asymmetrische Kryptographie.

Es gibt drei verschiedene Arten der Authentisierung für das TLS-Protokoll:

- Der Server weist sich per Zertifikat aus (siehe Server-Zertifikat [▶ 17])
- Client und Server weisen sich per Zertifikat aus (siehe Client/Server-Zertifikat [▶ 18])
- Pre-Shared-Keys (siehe Pre-Shared-Keys [▶ 18])

Über Vor- und Nachteile der verschiedenen Authentisierungsarten informieren Sie sich bitte in der Fachliteratur.



Beispielhafte Erläuterung anhand von RSA

Alle mit * gekennzeichneten Nachrichten sind optional und werden nicht zwingend benötigt. Die nachfolgenden Schritte beziehen sich auf das RSA-Verfahren und haben keine allgemeine Gültigkeit für andere Verfahren.

Client Hello: Der Client initiiert einen Verbindungsaufbau zum Server. Dabei werden unter anderem die verwendete TLS-Version, eine Zufallsfolge von Bytes (Client Random) und die vom Client unterstützten Cipher Suites übertragen.

Server Hello: Der Server sucht sich aus den vom Client offerierten Cipher Suites eine aus und legt diese für die Kommunikation fest. Besteht keine Schnittmenge zwischen den vom Client und Server unterstützten Cipher Suites, wird der TLS-Verbindungsaufbau abgebrochen. Zusätzlich kommuniziert auch der Server eine Zufallsfolge von Bytes (Server Random).

Certificate: Der Server präsentiert dem Client sein Zertifikat, damit der Client verifizieren kann, dass der Server auch der erwartete Server ist. Vertraut der Client dem Server-Zertifikat nicht, wird der TLS-Verbindungsaufbau abgebrochen. Das Server-Zertifikat enthält zusätzlich den öffentlichen Schlüssel des Servers.

(Server Key Exchange): Bei bestimmten Schlüsselaustauschalgorithmen reichen die Informationen aus dem Zertifikat nicht aus, damit der Client das sogenannte Pre-Master Secret erzeugen kann. In diesem Fall werden die fehlenden Informationen mittels Server Key Exchange übertragen.

Certificate Request: Der Server fordert vom Client ein Zertifikat an, um die Identität des Clients verifizieren zu können.

Server Hello Done: Der Server teilt dem Client mit, dass sein Versenden der initialen Informationen beendet ist.

Certificate: Der Client kommuniziert sein Zertifikat inklusive des öffentlichen Schlüssels an den Server. Der Ablauf ist der gleiche wie in entgegengesetzter Richtung: Vertraut der Server dem vom Client versendeten Zertifikat nicht, wird der Verbindungsaufbau abgebrochen.

Client Key Exchange: Der Client verwendet den öffentlichen Schlüssel des Servers, um durch asymmetrische Verschlüsselung ein von ihm generiertes Pre-Master Secret verschlüsselt an den Server zu schicken. Aus diesem Pre-Master Secret, dem Server Random und dem Client Random wird dann der symmetrische Schlüssel berechnet, der nach dem Verbindungsaufbau für die Kommunikation verwendet wird.

Certificate Verify: Der Client signiert die bisherigen Nachrichten des Handshakes mit seinem privaten Schlüssel. Da der Server den öffentlichen Schlüssel des Clients durch das Versenden des Zertifikats erhalten hat, kann er verifizieren, dass das präsentierte Zertifikat auch wirklich dem Client „gehört“.

Change Cipher Spec: Der Client teilt dem Server mit, dass er auf symmetrische Kryptographie umsteigt. Jede Nachricht vom Client an den Server ab hier ist signiert und verschlüsselt.

Finished: Der Client teilt dem Server verschlüsselt mit, dass der TLS-Verbindungsaufbau auf seiner Seite beendet ist. Die Nachricht enthält einen Hash und einen MAC über die bisherigen Nachrichten des Handshakes.

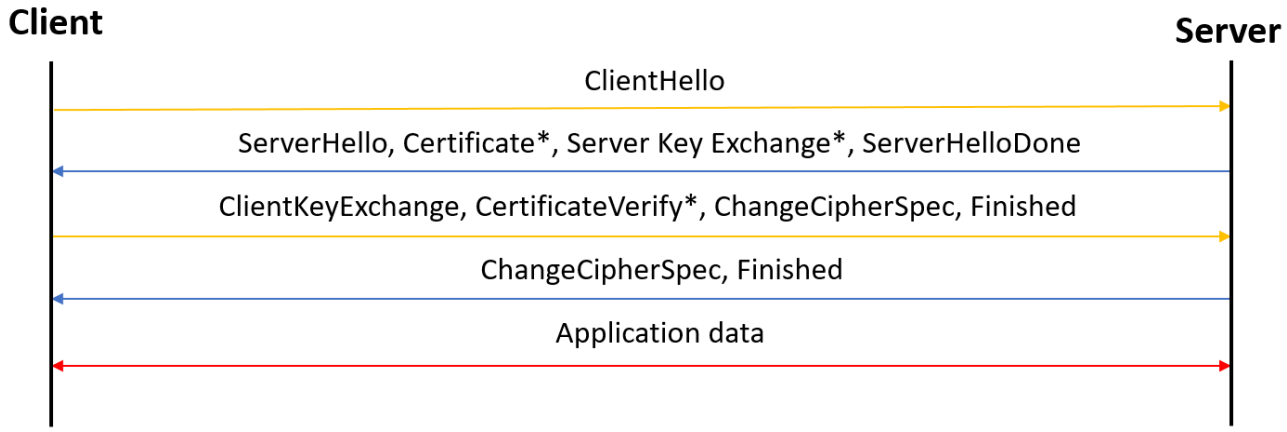
Change Cipher Spec: Der Server entschlüsselt das Pre-Master Secret, das der Client mit seinem öffentlichen Schlüssel verschlüsselt hat. Da der Server der Einzige ist, der seinen privaten Schlüssel besitzt, kann nur der Server dieses Pre-Master Secret entschlüsseln. So wird sichergestellt, dass der symmetrische Schlüssel nur Client und Server bekannt ist. Anschließend berechnet auch der Server den symmetrischen Schlüssel aus Pre-Master Secret und den beiden Zufallsfolgen und teilt dem Client mit, dass auch er jetzt mit symmetrischer Kryptographie kommuniziert. Jede Nachricht vom Server an den Client ab hier ist signiert und verschlüsselt. Durch die Generierung des symmetrischen Schlüssels kann der Server die Finished-Nachricht des Clients entschlüsseln und sowohl Hash als auch MAC verifizieren. Sollte diese Verifizierung fehlschlagen, wird die Verbindung abgebrochen.

Finished: Der Server teilt dem Client mit, dass der TLS-Verbindungsaufbau auf seiner Seite ebenfalls beendet ist. Die Nachricht enthält so wie beim Client einen Hash und einen MAC über die bisherigen Nachrichten des Handshakes. Auf der Seite des Clients wird dann dieselbe Verifikation vollzogen wie beim Server, auch hier gilt: Wenn Hash und MAC nicht erfolgreich entschlüsselt werden, wird die Verbindung abgebrochen.

Application Data: Client und Server kommunizieren nach Abschluss des TLS-Verbindungsaufbaus mit symmetrischer Kryptographie.

4.5.1.1 Server-Zertifikat

Dieser Abschnitt behandelt den Fall, dass nur der Client das Server-Zertifikat verifizieren will, der Server aber das Client-Zertifikat nicht. Der Kommunikationsablauf aus dem Kapitel Transportebene [▶ 15] verkürzt sich zu folgendem Ablauf.



Verifikation des Server-Zertifikats

Das Server-Zertifikat wird auf Client-Seite verifiziert. Dabei wird überprüft, ob es von einer bestimmten Certificate Authority signiert ist. Ist das nicht der Fall, wird die Verbindung vom Client abgebrochen, da er dem Server dann nicht vertraut.

Verwendung in TwinCAT

In TwinCAT wird in dem Fall der Dateipfad zum CA-Zertifikat angegeben (.PEM oder .DER-Datei) oder der Inhalt der .PEM-Datei als String. Im IoT-Treiber wird dann das vom Server präsentierte Zertifikat überprüft. Wenn die Zertifikatskette nicht von der angegebenen CA signiert ist, wird die Verbindung zum Server abgebrochen. Der nachfolgende Code stellt nur exemplarisch die beschriebenen Verbindungsparameter dar. Der Beispielcode bezieht sich sowohl auf den HTTP- als auch auf den MQTT-Client, exemplarisch wird der HTTP-Client verwendet.

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

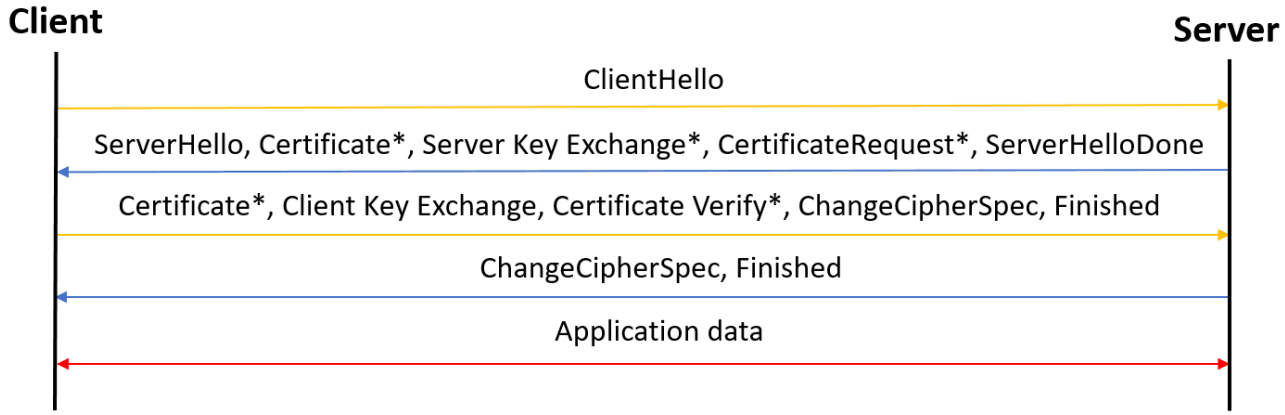
```
fbClient.stTLS.sCA:= 'C:\TwinCAT\3.1\Config\Certificates\someCA.pem';
```

Hat der Benutzer das CA-Zertifikat nicht zur Verfügung, kann trotzdem eine Verbindung hergestellt werden. Dazu gibt es eine boolsche Variable, mit der TwinCAT die Verifikation des Server-Zertifikats verboten wird. Die Verbindung wird zwar mit dem öffentlichen Schlüssel des unverifizierten Server-Zertifikats verschlüsselt hergestellt, ist aber anfälliger gegen Man-in-the-middle-Attacken.

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

4.5.1.2 Client/Server-Zertifikat

In diesem Abschnitt wird der Fall betrachtet, dass sowohl Client- als auch Server-Zertifikat verifiziert werden. Der im Vergleich zum Server-Zertifikat [▶ 17]-Kapitel leicht abgeänderte Kommunikationsablauf wird in der folgenden Grafik visualisiert. Die einzelnen Schritte des TLS-Verbindungsaufbaus sind im Kapitel Transportebene [▶ 15] beschrieben.



Verwendung in TwinCAT

In TwinCAT wird im Falle der Verwendung eines Client-Zertifikats ebenso wie beim CA-Zertifikat der Dateipfad (.PEM- oder .DER-Datei) oder der Inhalt der .PEM-Datei als String übergeben. Dieses Zertifikat präsentiert TwinCAT als Client dann dem Server. Für das Certificate Verify muss zusätzlich der private Schlüssel des Clients referenziert werden. Optional kann im Falle eines Passwortschutzes für den privaten Schlüssel auch dieses Passwort übergeben werden. Der Beispielcode bezieht sich sowohl auf den HTTP- als auch auf den MQTT-Client, exemplarisch wird der HTTP-Client verwendet.

```
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';
```

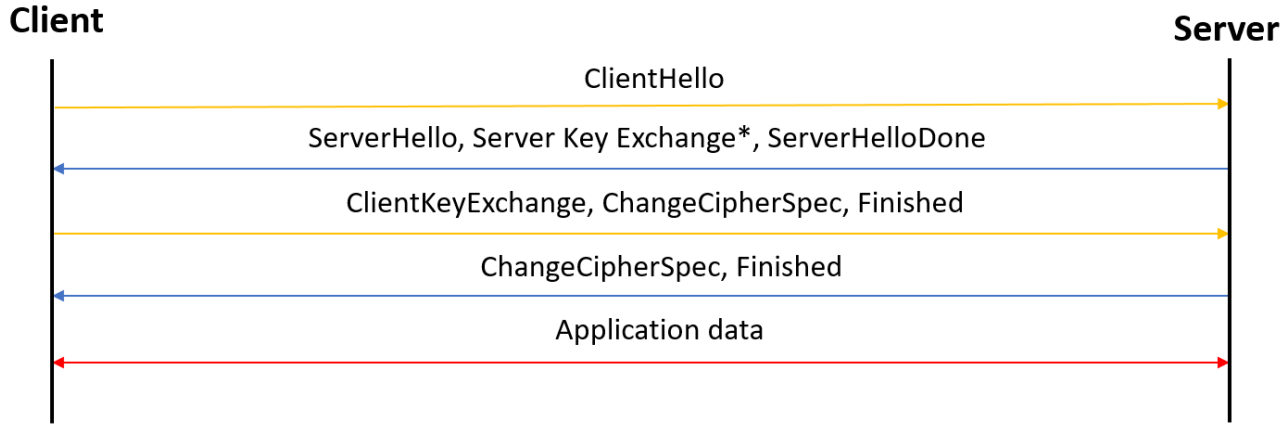
In dem Fall, dass ein Client-Zertifikat gesetzt ist, muss auch ein CA-Zertifikat zur Validierung des Server-Zertifikats gesetzt werden. Das ist durch das Verhalten des im IoT-Treiber verwendeten Security-Frameworks bedingt.

Wenn in diesem Fall die Validierung des Server-Zertifikats abgeschaltet werden soll, kann zusätzlich ein Flag zum Überspringen der Validierung gesetzt werden. Es ist aber nicht möglich, das CA-Zertifikat wegzulassen.

4.5.1.3 Pre-Shared-Keys

Standardmäßig werden beim TLS-Verbindungsaufbau asymmetrische Schlüsselpaare verwendet. Asymmetrische Kryptographie verbraucht höhere Rechenleistung, daher kann es in Umgebungen mit wenig CPU-Leistung eine Möglichkeit sein, Pre-Shared Keys (PSK) zu verwenden. Pre-Shared-Keys sind vorher geteilte, symmetrische Schlüssel.

Verglichen mit dem Kommunikationsablauf bei asymmetrischer Verschlüsselung fällt bei Verwendung von PSK das Zertifikat weg. Client und Server müssen sich über die sogenannte Identität auf einen PSK einigen. Der PSK ist per Definition vorher beiden Parteien bekannt.



Server Key Exchange: In dieser optionalen Nachricht kann der Server dem Client einen Hinweis auf die Identität des verwendeten PSK geben.

Client Key Exchange: Der Client gibt die Identity des PSK an, mit dem die Verschlüsselung durchgeführt werden soll.

Verwendung in TwinCAT

In TwinCAT wird die Identität des PSK als String angegeben, der PSK selbst wird als ByteArray in der Steuerung abgespeichert. Zusätzlich wird noch die Länge des PSK angegeben. Der Beispielcode bezieht sich sowohl auf den HTTP- als auch auf den MQTT-Client, exemplarisch wird der HTTP-Client verwendet.

```
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 Applikationsebene

Auch auf der Applikationsebene bieten sich verschiedene Sicherheits-Mechanismen an. Im Folgenden werden mehrere solcher Sicherheitsmechanismen beschrieben.

4.5.2.1 JSON Web Token (JWT)

JSON Web Token (JWT) sind ein offener Standard (nach RFC 7519), welche ein kompaktes und sich selbst beschreibendes Format definieren, um Informationen sicher zwischen Kommunikationsteilnehmern in Form eines JSON Objekts zu übertragen. Die Authentizität der übertragenen Information kann hierbei verifiziert und sichergestellt werden, da ein JWT mit einer digitalen Signatur versehen wird. Die Signatur kann hierbei über ein Shared Secret (via HMAC Algorithmus) oder einen Public/Private Key (via RSA) erfolgen.

Das am weitesten verbreitete Anwendungsbeispiel für JWT ist die Autorisierung eines Geräts oder Benutzers an einem Service. Sobald sich ein Benutzer an dem Service angemeldet hat, beinhalten alle weiteren Anfragen an den Service das JWT. Anhand des JWT kann der Service dann entscheiden, auf welche weiteren Dienste oder Ressourcen der Benutzer zugreifen darf. Hierdurch können zum Beispiel Single Sign On Lösungen in Cloud-Diensten realisiert werden.

Die SPS Bibliothek Tc3_JsonXml stellt über die Methode FB_JwtEncode die Möglichkeit bereit ein JWT zu erzeugen und signieren.

4.5.2.2 AWS Signature Version 4

AWS Signature V4 fügt Anfragen an AWS-Services (Bsp.: Die Instanziierung einer virtuellen Maschine über den EC2-Service), die über das HTTP-Protokoll versendet werden, Authentisierungsinformationen hinzu. Aus Sicherheitsgründen müssen die meisten Anfragen an AWS-Services mit einem Zugriffsschlüssel signiert werden. Dieser Zugriffsschlüssel, bestehend aus „access key ID“ und „secret access key“, kann in einem entsprechenden AWS-Account eingerichtet und verwaltet werden. Genauere Informationen entnehmen Sie bitte der AWS-Dokumentation.

4.6 Neuparametrierung

In bestimmten Fällen kann eine Neuparametrierung des HTTP-Clients während des laufenden Betriebes durch einen Online-Change notwendig sein. Dabei kann diese Neuparametrierung entweder automatisch im Programmablauf erfolgen oder bei Bedarf manuell getriggert werden. Dies ist aber von der Implementierung abhängig.

Beispielhafte Anwendungsfälle für eine Neuparametrierung können der Austausch eines in Kürze ablaufenden Tokens, zu erneuernde Zertifikate oder eine geänderte IP-Adresse des HTTP-Servers sein. Im Folgenden wird der notwendige Ablauf für eine Neuparametrierung eines bereits parametrierten HTTP-Clients beschrieben.

Für die zyklische Hintergrundkommunikation eines HTTP-Client-Bausteins mit dem TwinCAT-Treiber muss im Hintergrund zyklisch die Execute-Methode aufgerufen werden (siehe FB_IotHttpClient [▶ 23]). Bei einem Programmstart wird durch den Aufruf dieser Methode zunächst die Parametrierung der HTTP-Client-Instanz vorgenommen, nach deren Abschluss das Senden von HTTP-Requests möglich ist. Der Abschluss dieser Parametrierung kann durch die Variable bConfigured erkannt werden. Sobald die Variable den Wert TRUE hat, ist der HTTP-Client vollständig konfiguriert.

Um eine Neuparametrierung vorzunehmen, muss der Aufruf der Execute-Methode ausgesetzt und die Disconnect-Methode aufgerufen werden. Dabei ist es wichtig zu beachten, dass zum Zeitpunkt des Aufrufs keine HTTP-Requests mehr abgesendet werden. Durch den Aufruf der Disconnect-Methode wird bConfigured auf FALSE gesetzt. Anschließend werden die Parameter neu gesetzt und die Execute-Methode dann wieder wie gewohnt aufgerufen. Das folgende Code-Snippet zeigt eine einfache Neuparametrierung anhand eines Triggers.

```
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
```

Durch die Trigger-Variable bReset wird der Aufruf der Execute-Methode ausgesetzt und die Disconnect-Methode aufgerufen. Sobald bConfigured am HTTP-Client-Baustein auf FALSE gesetzt ist, werden die Parameter für den Hostnamen und das CA-Zertifikat neu gesetzt. Anschließend wird bReset wieder zurückgesetzt und somit die Excute-Methode wieder aufgerufen.

4.7 URL-Redirects

Mit einem Server-seitigen URL-Redirect zeigt ein Server einem Client, dass eine angefragte Ressource auf dem Server entweder permanent oder temporär verschoben ist. Im IoT-Treiber werden solche Redirects nicht automatisch ausgewertet. Es bleibt also der Applikationslogik überlassen, einen URL-Redirect auszuwerten.

Beispielsweise sei hier der HTTP-Statuscode 301 („Moved permanently“) zu nennen. Hier teilt ein Server einem Client mit, dass eine Ressource auf dem Server dauerhaft verschoben wurde. Um dem Client aber zusätzlich auch die neue Adresse der Ressource mitteilen zu können, wird der HTTP-Response ein Header-Field mit dem Namen „Location“ hinzugefügt. In diesem Header-Field steht dann die neue URL, an der sich die Ressource befindet.

In einer TwinCAT-Applikation könnte nach dem Empfangen einer HTTP-Response über die Abfrage des Status-Codes eine Applikationslogik implementiert werden, die aus einer Response mit Statuscode 301 die neue URL aus der Response extrahiert und eine erneute Anfrage mit der aktualisierten URL an den Server schickt. Dazu muss das „Location“-Header-Field der Response abgefragt und als Ziel-URL für einen erneuten Request verwendet werden.

4.8 Cookies

Cookies werden im HTTP-Umfeld für drei Use-Cases eingesetzt. Zu den Use-Cases gehört zum einen Session-Management, was beispielsweise für Logins genutzt wird oder allgemein für alles, was der Server sich „merken“ muss. Zum anderen werden Cookies aber auch für Personalisierung und Tracking eingesetzt.

Aus technischer Sicht sendet ein HTTP-Server bei einer Antwort an den Client ein oder mehrere Header-Fields mit dem Namen „Set-Cookie“ in der Response zurück. Diese Cookies werden dann in zukünftigen Anfragen an den HTTP-Server mitgeschickt, in diesem Fall aber mit einem Semikolon separiert in einem einzelnen Header-Field.

In TwinCAT werden die oben beschriebenen Header-Fields mit den Cookies automatisch zu einem Header-Field zusammengefasst und durch einen Zeilenvorschub getrennt. Dies gilt nicht nur für Header-Fields vom Typ „Set-Cookie“, sondern immer dann, wenn eine HTTP-Response des Servers mehrere gleichnamige Header-Fields enthält. Folgendes Beispiel soll zur Veranschaulichung dienen:

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

Wird diese Response vom TwinCAT IoT-Treiber empfangen, werden die beiden Header-Fields mit dem Namen „Set-Cookie“ automatisch zu einem zusammengefügt. Folgender String wird dann nach Abfrage des Header-Fields im Programmcode ausgegeben:

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

5 PLC API

5.1 Tc3_IotBase

5.1.1 FB_IotHttpAwsSigV4HeaderFieldMap

Dieser Funktionsbaustein ermöglicht das Signieren eines Headers mit AWS Signature Version 4 (siehe AWS Signature Version 4 [▶ 20]).

Syntax

Definition:

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

Ausgänge

Name	Typ	Beschreibung
bError	BOOL	Wird TRUE, sobald eine Fehlersituation auftritt
hrErrorCode	HRESULT	Gibt einen ADS Return Code zurück. Eine Erläuterung zu den möglichen ADS Return Codes befindet sich im Anhang.

Methoden

Name	Beschreibung
AddField [▶ 25]	Fügt einer Instanz dieses Funktionsbausteins ein Header-Feld hinzu.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4024.12 oder höher	IPC oder CX (x86, x64, ARM)	Tc3_IotBase

5.1.1.1 SetParameter

Diese Methode setzt die benötigten Parameter an einem für Anfragen an AWS-Services gedachten Header und signiert diesen. Die einzelnen Parameter müssen für die Signierung alphabetisch sortiert werden. Dies wird vom Treiber intern gemacht, der Benutzer muss folglich nicht auf die Reihenfolge achten.

Syntax

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

Eingänge

Name	Typ	Beschreibung
service	STRING	Code des AWS-Services
region	STRING	Code des AWS-Rechenzentrums.
accessKey	STRING	Der AccessKey des zweiteiligen Zugriffsschlüssels eines AWS IAM-Benutzers. Wird zur Authentifizierung von Anfragen benötigt.
secretKey	STRING	Der SecretKey des zweiteiligen Zugriffsschlüssels eines AWS IAM-Benutzers. Wird zur Authentifizierung von Anfragen benötigt.
signedHeaders	STRING	Beschreibt eine durch Semikolons getrennte Liste von HTTPS-Headern, die in die Signatur aufgenommen werden sollen. Weiteres entnehmen Sie bitte der AWS-Dokumentation.

Rückgabewert

Name	Typ	Beschreibung
SetParameter	BOOL	Gibt TRUE zurück, wenn der Methodenaufruf erfolgreich war.

5.1.2 FB_IotHttpClient

Dieser Funktionsbaustein ermöglicht die Kommunikation mit einem HTTP-Server.

Ein Client-Funktionsbaustein ist für die Verbindung mit genau einem Server verantwortlich. Die Methode Execute() des Funktionsbausteins muss zyklisch aufgerufen werden, um die Funktionalität des HTTP-Clients zu aktivieren. Die genaue Ressource auf dem Server wird mit der Eigenschaft sUri des Funktionsbausteins FB_IotHttpRequest spezifiziert.

Alle Verbindungsparameter sind als Eingangsparameter vorhanden und werden ausgewertet, wenn eine Verbindung hergestellt wird.

Syntax

Definition:

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

Eingänge

Name	Typ	Beschreibung
sHostName	STRING (255)	sHostName kann als Name oder als IP-Adresse festgelegt werden. Wenn keine Angaben gemacht werden, wird der lokale Host verwendet.
nHostPort	UINT	Hier kann der Hostport festgelegt werden. Der Standardwert ist 80.
bKeepAlive	BOOL	Legt fest, ob die Verbindung zum Server aufrechterhalten werden soll, bis entweder der Server oder Client sie beendet. Beachten Sie, dass nicht jeder Server diese Funktion unterstützt.
tConnectionTimeout	TIME	Legt das Verbindungs-Timeout fest.
stTLS	ST_IotSocketTls [▶ 31]	Wenn der Server eine TLS-gesicherte Verbindung anbietet, kann die erforderliche Konfiguration hier implementiert werden.

Ausgänge

Name	Typ	Beschreibung
bError	BOOL	Wird TRUE, sobald eine Fehlersituation auftritt.
hrErrorCode	HRESULT	Gibt einen ADS Return Code zurück. Eine Erläuterung zu den möglichen ADS Return Codes befindet sich im Anhang.
bConfigured	BOOL	Wird TRUE, wenn die initiale Konfiguration abgeschlossen ist.
bConnected	BOOL	Wird TRUE, wenn eine Verbindung zum Host hergestellt ist. Beachten Sie, dass die Verbindung in der Regel nach jeder Anfrage geschlossen wird, abhängig vom Eingangsparameter bKeepAlive.

Methoden

Name	Beschreibung
Disconnect	Trennt den Client vom Server, wenn eine aktive Verbindung besteht.
Execute	Methode für Hintergrundkommunikation mit dem TwinCAT-Treiber. Die Methode muss für jede Funktionsbausteininstanz zyklisch aufgerufen werden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4024.7 oder höher	IPC oder CX (x86, x64, ARM)	Tc3_IotBase

5.1.3 FB_IotHttpHeaderFieldMap

Dieser Funktionsbaustein bietet die Möglichkeit, einer HTTP-Anfrage zusätzliche Header-Felder hinzuzufügen. Hierzu muss die Methode AddField an diesem Funktionsbaustein aufgerufen werden. Anschließend muss der Funktionsbaustein an den HTTP-Befehl weitergeleitet werden, der an den Server gesendet wird.

Syntax

Definition:

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

Ausgänge

Name	Typ	Beschreibung
bError	BOOL	Wird TRUE, sobald eine Fehlersituation auftritt.
hrErrorCode	HRESULT	Gibt einen ADS Return Code zurück. Eine Erläuterung zu den möglichen ADS Return Codes befindet sich im Anhang.

Methoden

Name	Beschreibung
AddField [▶ 25]	Fügt einer Instanz dieses Funktionsbausteins ein Header-Feld hinzu.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4024.7 oder höher	IPC oder CX (x86, x64, ARM)	Tc3_IotBase

5.1.3.1 AddField

Diese Methode fügt einer Instanz des Funktionsbausteins FB_IotHttpHeaderFieldMap ein zusätzliches Header-Feld hinzu.

Syntax

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

Eingänge

Name	Typ	Beschreibung
sField	STRING	Name des Feldes, das dem Header der HTTP-Anfrage hinzugefügt wird.
sValue	STRING	Wert des hinzugefügten Header-Feldes.
bAppendValue	BOOL	Ersetzt den vorhandenen Header-Feldwert (FALSE) oder hängt den Wert an den vorhandenen Wert an (TRUE).

Rückgabewert

Name	Typ	Beschreibung
AddField	BOOL	Gibt TRUE zurück, wenn der Methodenaufruf erfolgreich war.

5.1.3.2 FindField

Diese Methode sucht in einer Instanz des Funktionsbausteins FB_IotHttpHeaderFieldMap nach einem bestimmten Header-Field.

Syntax

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

Eingänge

Name	Typ	Beschreibung
sField	STRING	Name des Feldes, das in dem Header gefunden werden soll.
pValue	PVOID	Zeiger auf den Datenspeicher, wo das gefundene Header-Field hingespeichert werden soll.
nValueSize	UDINT	Die Größe der Speichervariablen.

Rückgabewert

Name	Typ	Beschreibung
FindField	BOOL	Gibt TRUE zurück, wenn der Methodenaufruf erfolgreich war.

5.1.4 FB_IotHttpRequest

Dieser Funktionsbaustein ermöglicht es der SPS, HTTP-Befehle an den Server zu senden, der mit dem Funktionsbaustein FB_IotHttpClient konfiguriert worden ist. Um eine HTTP-Antwort verarbeiten zu können, muss der Ausgang bBusy dieses Funktionsbausteins zuerst FALSE zurückgeben. Dann können die Inhaltsnutzlast, bestimmte Header-Felder oder ein JSON-String aus der Antwort extrahiert werden.

Syntax

Definition:

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

```
    nStatusCode      : UINT;
    nContentSize     : UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
sContentType	STRING	Inhaltstyp der Anfrage, z. B. der Initialwert „text/html; charset=UTF-8“.
eCompressionMode	E_IotHttpCompressionMode	Aufzählung der verschiedenen Kompressions-Modi. Weitere Informationen werden im Kapitel Kompression [▶ 14] zur Verfügung gestellt. Die Aufzählung befindet sich im Anhang.

Ausgänge

Name	Typ	Beschreibung
bBusy	BOOL	Dieser Ausgang bleibt TRUE, bis der Funktionsbaustein den Befehl ausgeführt hat.
bError	BOOL	Wird TRUE, sobald eine Fehlersituation auftritt.
eErrorId	ETcIotHttpRequestError	Aufzählung des Statuscodes auf Transportebene, z. B. ob eine Zertifikatsvalidierung fehlgeschlagen ist. Die Aufzählung befindet sich im Anhang.
nStatusCode	UINT	HTTP-Statuscode, der vom Server in der Antwort gesendet wird.
nContentSize	UDINT	Größe der Inhaltsnutzlast.

Methoden

Name	Beschreibung
GetContent [▶ 28]	Liest die Antwort vom HTTP-Server.
GetHeaderField [▶ 28]	Liest den Wert eines bestimmten Header-Feldes.
GetJsonDomContent [▶ 29]	Parst die Serverantwort als JSON-Objekt.
SendJsonDomRequest [▶ 29]	Sendet eine Anfrage an den Server als JSON-Objekt.
SendRequest [▶ 30]	Sendet einen HTTP-Befehl an den Server.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4024.7 oder höher	IPC oder CX (x86, x64, ARM)	Tc3_IotBase

5.1.4.1 GetContent

Diese Methode liest die HTTP-Antwort und speichert sie in eine Variable. Um den Inhalt aus einer HTTP-Antwort auslesen zu können, muss der Ausgang bBusy des Funktionsbausteins FB_IotHttpRequest FALSE zurückgeben.

Syntax

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

Eingänge

Name	Typ	Beschreibung
pContent	PVOID	Zeiger auf den Datenspeicher, wo die gelesenen Daten gespeichert werden sollen.
nContentSize	UDINT	Größe der Inhaltsnutzlast.
bSetNullTermination	BOOL	Legt fest, ob die Nutzlast mit Null-Terminierung oder ohne Null-Terminierung gelesen werden soll.

Rückgabewert

Name	Typ	Beschreibung
GetContent	BOOL	Gibt TRUE zurück, wenn der Methodenaufruf erfolgreich war.

5.1.4.2 GetHeaderField

Diese Methode kann den Wert eines bestimmten Header-Feldes aus einer HTTP-Antwort lesen. Um ein Header-Feld aus einer HTTP-Antwort auslesen zu können, muss der Ausgang bBusy des Funktionsbausteins FB_IotHttpRequest FALSE zurückgeben.

Syntax

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

Eingänge

Name	Typ	Beschreibung
sField	STRING	Name des bestimmten Header-Feldes.
pValue	PVOID	Zeiger auf den Datenspeicher, wo die gelesenen Daten gespeichert werden sollen.
nValueSize	UDINT	Die maximale Größe, die der Puffer dieser Methode haben kann.

Rückgabewert

Name	Typ	Beschreibung
GetHeaderField	BOOL	Gibt TRUE zurück, wenn der Methodenaufruf erfolgreich war.

5.1.4.3 GetJsonDomContent

Diese Methode parst die Antwort in ein JSON-Objekt. Wenn der Server immer mit einem JSON-Objekt antwortet, wird der Schritt der Umwandlung eines Strings in ein JSON-Objekt übersprungen. Um den Inhalt einer HTTP-Antwort auslesen zu können, muss der Ausgang bBusy des Funktionsbausteins FB_IotHttpRequest FALSE zurückgeben.

Syntax

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

Eingänge

Name	Typ	Beschreibung
fbJson	REFERENCE TO FB_JsonDomParser	Die Instanz des JsonDomParser, die zum Parsen des JSON-Objekts verwendet werden soll.

Rückgabewert

Name	Typ	Beschreibung
GetJsonDomContent	SJsonValue	JSON-Wurzelknoten, um das Parsen zu starten.

5.1.4.4 SendJsonDomRequest

Diese Methode sendet eine Anfrage mit einer JSON-Objektnutzlast an einen HTTP-Server. Diese Methode kann z. B. verwendet werden, wenn der Server nur JSON-Objektnutzlasten akzeptiert.

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
```

Eingänge

Name	Typ	Beschreibung
sUri	STRING	Die Kennung der Ressource auf dem Host. Beachten Sie, dass die sUri an den Hostnamen des IotHttpClient angehängt wird.
fbClient	REFERENCE TO FB_IotHttpClient	Die Instanz des HTTP-Client-Funktionsbausteins, von der die Anfrage gesendet werden soll.
eRequestType	ETcIotHttpRequestType	Der Typ des HTTP-Befehls, den die Anfrage haben sollte. Die Aufzählung befindet sich im Anhang.
fbJson	REFERENCE TO FB_JsonDomParser	Die Instanz von JsonDomParser zum Parsen eines Strings in ein JSON-Objekt.
fbHeader	REFERENCE TO FB_IotHttpHeaderFieldMap	Die Instanz von IotHttpHeaderFieldMap für die Bearbeitung des Headers der HTTP-Anfrage.

Rückgabewert

Name	Typ	Beschreibung
SendJsonDomRequest	BOOL	Gibt TRUE zurück, wenn der Methodenaufruf erfolgreich war.

5.1.4.5 SendRequest

Diese Methode sendet eine Anfrage an einen HTTP-Server.

Syntax

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

Eingänge

Name	Typ	Beschreibung
sUri	STRING	Die Kennung der Ressource auf dem Host. Beachten Sie, dass die sUri an den Hostnamen des FB_IotHttpClient angehängt wird.
fbClient	REFERENCE TO FB_IotHttpClient	Die Instanz des HTTP-Client-Funktionsbausteins, von der die Anfrage gesendet werden soll.
eRequestType	ETcIotHttpRequestType	Der Typ des HTTP-Befehls, den die Anfrage haben sollte. Die Aufzählung befindet sich im Anhang.
pContent	PVOID	Zeiger auf den Datenspeicher, von dem die Anfragenutzlast gelesen werden soll.
nContentSize	UDINT	Größe der Anfragenutzlast.
fbHeader	REFERENCE TO FB_IotHttpHeaderFieldMap	Die Instanz des FB_IotHttpHeaderFieldMap für die Bearbeitung des Headers der HTTP-Anfrage.

Rückgabewert

Name	Typ	Beschreibung
SendRequest	BOOL	Gibt TRUE zurück, wenn der Methodenaufruf erfolgreich war.

5.1.5 ST_IotSocketTls

Der folgende Typ enthält die TLS-Sicherheitseinstellungen für den HTTP-Client.
Entweder CA (Zertifizierungsstelle) oder PSK (PreSharedKey) können verwendet werden.

Syntax

Definition:

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

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

Parameter

Name	Typ	Beschreibung
sCA	STRING(255)	Zertifikat der Certificate Authority (CA)
sCert	STRING(255)	Client-Zertifikat für die Authentifizierung beim Server
sKeyFile	STRING(255)	Private Key des Clients
sKeyPwd	STRING(255)	Passwort des Private Keys, falls anwendbar
sCrl	STRING(255)	Pfad zur Certificate Revocation List, welche im Format PEM oder DER vorliegen kann
sCiphers	STRING(255)	Zu verwendende Ciper Suites, angegeben im OpenSSL-String-Format
sVersion	STRING(80)	Zu verwendende TLS-Version
bNoServerCertCheck	BOOL	Deaktiviert die Überprüfung des Server-Zertifikats auf Gültigkeit
sPskIdentity	STRING(255)	PreSharedKey-Identität für TLS-PSK Verbindung
aPskKey	ARRAY[1..64] OF BYTE	PreSharedKey für TLS-PSK-Verbindung
nPskKeyLen	USINT	Länge des PreSharedKey in Bytes

Alle Strings und Arrays, die mit einem * gekennzeichnet worden sind, werden mit dem Wert in Klammern initialisiert. Über die Parameterliste kann auf diese Werte zugegriffen werden und können diese geändert werden. Dies ist nicht während der Laufzeit, sondern nur vor der Kompilierung des Codes möglich.

5.1.6 Parameterliste

Die Parameterliste der Bibliothek Tc3_IotBase kann verwendet werden, um die Größe verschiedener Parameter zu ändern. Diese Änderung ist nicht während der Laufzeit möglich und kann nur vor der Kompilierung des Codes vorgenommen werden. Für einen besseren Überblick werden in dieser Dokumentation nur die Parameter aufgelistet, die für HTTP wichtig sind.

Syntax

Definition:

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

Parameter

Name	Typ	Beschreibung
cSizeOfHttpContentType	UDINT(32..10000)	Stringgrößendefinition von HTTP-Inhaltstyp (in Bytes).
cSizeOfHttpClientHostName	UDINT(32..10000)	Stringgrößendefinition von HTTP-Client-Hostname (in Bytes).
cSizeOfHttpTlsFilePaths	UDINT(32..10000)	Stringgrößendefinition von HTTP-TLS-Dateipfad (in Bytes).
cSizeOfHttpTlsKeyFile	UDINT(32..10000)	Stringgrößendefinition von HTTP-TLS-Schlüsseldatei (in Bytes).
cSizeOfHttpTlsKeyPwd	UDINT(32..10000)	Stringgrößendefinition von HTTP-TLS-Passwort des Private Key (in Bytes).
cSizeOfHttpTlsCiphers	UDINT(32..10000)	Stringgrößendefinition von HTTP-TLS-Cipher-Suites (in Bytes).
cSizeOfHttpTlsPskIdentity	UDINT(32..10000)	Stringgrößendefinition von HTTP-TLS-PSK-Identität (in Bytes).
cSizeOfHttpTlsPskKey	UDINT(32..10000)	Stringgrößendefinition von HTTP-TLS-PSK (in Bytes).

5.2 Tc3_JsonXml

5.2.1 Funktionsbausteine

5.2.1.1 FB_JsonDomParser

Dieser Funktionsblock ist von demselben internen Funktionsbaustein abgeleitet wie der FB_JsonDynDomParser [▶ 63] und bietet somit dasselbe Interface.

Die beiden abgeleiteten Funktionsblöcke unterscheiden sich nur in ihrer internen Speicherverwaltung. Der FB_JsonDomParser ist optimiert für schnelles und effizientes Parsen und Erstellen von JSON-Dokumenten, die nur wenig verändert werden. Für JSON-Dokumente, an denen viele Änderungen (bspw. das zyklische Ändern eines bestimmten Wertes im JSON-Dokument) vorgenommen werden, wird der Funktionsbaustein FB_JsonDynDomParser [▶ 63] empfohlen.

⚠ WARNUNG

Verwendung von Router Speicher

Der Funktionsbaustein zieht bei jeder Änderung, z.B. bei den Methoden SetObject() oder SetJson(), neuen Speicher an. Die verwendete Menge von Router Speicher kann hierdurch nach wiederholten Aktionen stark anwachsen. Dieser allokierte Speicher wird erst durch den Aufruf der Methoden NewDocument [▶ 52]() oder ParseDocument [▶ 52]() wieder freigegeben.

Strings im UTF-8-Format

Die hier verwendeten Variablen vom Typ STRING nutzen das UTF-8-Format. Diese STRING-Formatierung ist üblich bei IoT/MQTT-Kommunikation sowie JSON-Dokumenten.

Um Sonderzeichen und Texte verschiedenster Sprachen empfangen zu können, wird der Zeichensatz in den Bibliotheken Tc3_IotBase und Tc3_JsonXml nicht auf den typischen Zeichensatz vom Datentyp STRING beschränkt. Stattdessen wird der Unicode-Zeichensatz als UTF-8-Format in Verbindung mit dem Datentyp STRING verwendet.

Bei Verwendung des ASCII-Zeichensatzes besteht kein Unterschied zwischen der typischen Formatierung in einem STRING und der UTF-8-Formatierung eines STRING.

Weitere Informationen zum UTF-8-STRING-Format sowie vorhandenen Anzeige- und Konvertierungsmöglichkeiten finden Sie in der Dokumentation der SPS-Bibliothek Tc2_Utilities.

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4022	x86, x64, ARM	Tc3_JsonXml

5.2.1.1.1 AddArrayMember

Diese Methode fügt ein Array-Member zu einem JSON-Objekt hinzu.

Syntax

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

Beispielaufruf:

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

5.2.1.1.2 AddBase64Member

Diese Methode fügt ein Base64-Member zu einem JSON-Objekt hinzu. Als Eingabeparameter kann z. B. eine Struktur adressiert werden. Die entsprechende Base64-Kodierung erfolgt durch die Methode.

Syntax

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

Beispielaufruf:

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

5.2.1.1.3 AddBoolMember

Diese Methode fügt ein Bool-Member zu einem JSON-Objekt hinzu.

Syntax

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

Beispielaufruf:

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

5.2.1.1.4 AddDateTimeMember

Diese Methode fügt ein DateTime-Member zu einem JSON-Objekt hinzu.

Syntax

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

Beispielaufruf:

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

5.2.1.1.5 AddDcTimeMember

Diese Methode fügt ein DcTime-Member zu einem JSON-Objekt hinzu.

Syntax

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

Beispielaufruf:

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

5.2.1.1.6 AddDoubleMember

Diese Methode fügt ein Double-Member zu einem JSON-Objekt hinzu.

Syntax

```
METHOD AddDoubleMember : SJsonValue
VAR_INPUT
  v     : SJsonValue;
  value : LREAL;
```

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

Beispielaufruf:

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

5.2.1.1.7 AddFileTimeMember

Diese Methode fügt ein FileTime-Member zu einem JSON-Objekt hinzu.

Syntax

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

Beispielaufruf:

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

5.2.1.1.8 AddHexBinaryMember

Diese Methode fügt ein HexBinary-Member zu einem JSON-Objekt hinzu.

Syntax

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

Beispielaufruf:

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

5.2.1.1.9 AddInt64Member

Diese Methode fügt ein Int64-Member zu einem JSON-Objekt hinzu.

Syntax

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

Beispielaufruf:

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

5.2.1.1.10 AddIntMember

Diese Methode fügt ein Int-Member zu einem JSON-Objekt hinzu.

Syntax

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

Beispielaufruf:

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

5.2.1.1.11 AddJsonMember

Diese Methode fügt ein JSON-Member zu einem JSON-Objekt hinzu.

Syntax

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

Beispielaufruf:

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

5.2.1.1.12 AddNullMember

Diese Methode fügt ein NULL-Member zu einem JSON-Objekt hinzu.

Syntax

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

Beispielaufruf:

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

5.2.1.1.13 AddObjectMember

Diese Methode fügt ein Object-Member zu einem JSON-Objekt hinzu.

Syntax

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

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

Beispielaufruf:

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

5.2.1.1.14 AddStringMember

Diese Methode fügt ein String-Member zu einem JSON-Objekt hinzu.

Syntax

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

Beispielaufruf:

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

5.2.1.1.15 AddUint64Member

Diese Methode fügt ein UInt64-Member zu einem JSON-Objekt hinzu.

Syntax

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

Beispielaufruf:

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

5.2.1.1.16 AddUintMember

Diese Methode fügt ein UInt-Member zu einem JSON-Objekt hinzu.

Syntax

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

Beispielaufruf:

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

5.2.1.1.17 ArrayBegin

Diese Methode liefert das erste Element eines Arrays und kann zusammen mit den Methoden ArrayEnd() und NextArray() zur Iteration durch ein JSON-Array verwendet werden.

Syntax

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

Beispielaufruf:

```
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

Diese Methode liefert das letzte Element eines Arrays und kann zusammen mit den Methoden ArrayBegin() und NextArray() zur Iteration durch ein JSON-Array verwendet werden.

Syntax

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

Beispielaufruf:

```
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

Diese Methode löscht den Inhalt eines Arrays.

Syntax

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

Beispielaufruf:

Gegeben sei das folgende JSON-Dokument, das in den DOM-Speicher geladen wird:

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

Die Werte des JSON-Arrays „array“ sollen gelöscht werden. Zunächst wird das JSON-Dokument iterativ nach dem Property „array“ durchsucht, anschließend werden alle Elemente des Arrays durch Aufruf der Methode ClearArray() gelöscht.

```
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

Diese Methode kopiert den Inhalt des DOM-Speichers in eine Variable vom Datentyp STRING, welche eine beliebige Länge haben kann. Als Rückgabewert liefert die Methode die Länge des Strings (inklusive Nullterminierung). Falls der Zielpuffer zu klein ist, wird dieser durch eine Nullterminierung geleert und als Länge 0 zurückgegeben.

Syntax

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

Beispielaufruf:

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

5.2.1.1.21 CopyJson

Diese Methode extrahiert ein JSON-Objekt aus einem Key und speichert dieses in einer Variablen vom Datentyp STRING. Dieser STRING kann eine beliebige Länge haben. Als Rückgabewert liefert die Methode die Länge des kopierten JSON-Objekts (inklusive Nullterminierung). Falls der Zielpuffer zu klein ist, wird dieser durch eine Nullterminierung geleert und als Länge 0 zurückgegeben.

Syntax

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

Beispielaufruf:

Gegeben sei das folgende JSON-Dokument, das in den DOM-Speicher geladen wird:

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

Der Wert des JSON-Objekts „meta“ soll extrahiert und in einer Variablen vom Datentyp STRING gespeichert werden. Zunächst wird das JSON-Dokument iterativ nach dem Property „meta“ durchsucht, anschließend wird dessen Wert bzw. Unterobjekt durch Aufruf der Methode CopyJson() extrahiert.

```
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
```

Die Variable sString hat nach diesem Durchlauf folgenden Inhalt:

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

5.2.1.1.22 CopyString

Diese Methode kopiert den Wert eines Keys in eine Variable vom Datentyp STRING, welche eine beliebige Länge haben kann. Als Rückgabewert liefert die Methode die Länge des kopierten Strings (inklusive Nullterminierung). Falls der Zielpuffer zu klein ist, wird dieser durch eine Nullterminierung geleert und als Länge 0 zurückgegeben.

Syntax

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

Beispielaufruf:

Gegeben sei das folgende JSON-Dokument, das in den DOM-Speicher geladen wird:

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

Der Wert des Keys „clickType“ soll extrahiert und in einer Variablen vom Datentyp STRING gespeichert werden. Zunächst wird das JSON-Dokument iterativ nach dem Property „clickType“ durchsucht.

```
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
```

Die Variable sString hat nach diesem Durchlauf folgenden Inhalt:

```
SINGLE
```

5.2.1.1.23 FindMember

Diese Methode sucht in einem JSON-Dokument nach einem bestimmten Property und gibt dieses zurück. Wenn kein entsprechendes Property gefunden wird, wird 0 zurückgegeben.

Syntax

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

Beispielaufruf:

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

5.2.1.1.24 FindMemberPath

Diese Methode sucht in einem JSON-Dokument nach einem bestimmten Property und gibt dieses zurück. Das Property wird hierbei nach dessen Pfad im Dokument spezifiziert. Wenn kein entsprechendes Property gefunden wird, wird 0 zurückgegeben.

Syntax

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

Beispielaufruf:

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

5.2.1.1.25 GetArraySize

Diese Methode liefert die Anzahl der Elemente in einem JSON-Array.

Syntax

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

Beispielaufruf:

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

5.2.1.1.26 GetArrayValue

Diese Methode liefert den Wert an der aktuellen Iterator-Position eines Arrays.

Syntax

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

Beispielaufruf:

```
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

Diese Methode liefert den Wert eines Arrays an einem angegebenen Index.

Syntax

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

Beispielaufruf:

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

5.2.1.1.28 GetBase64

Diese Methode dekodiert einen Base64-Wert aus einem JSON-Property. Wenn sich hinter dem Base64-Wert z. B. der Inhalt einer Datenstruktur befindet, kann der dekodierte Inhalt auch wieder auf eine identische Datenstruktur gelegt werden.

Syntax

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

Beispielaufruf:

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

5.2.1.1.29 GetBool

Diese Methode liefert den Value eines Properties vom Datentyp BOOL.

Syntax

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

5.2.1.1.30 GetDateTime

Diese Methode liefert den Value eines Properties vom Datentyp DATE_AND_TIME.

Syntax

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

5.2.1.1.31 GetDcTime

Diese Methode liefert den Values eines Properties vom Datentyp DCTIME.

Syntax

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

Beispielaufruf:

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

5.2.1.1.32 GetDocument

Diese Methode gibt den Inhalt des DOM-Speichers als Datentyp STRING(255) zurück. Bei längeren Zeichenfolgen wird von der Methode ein NULL-String zurückgegeben. In diesem Fall muss die Methode CopyDocument [▶ 40]() verwendet werden.

Syntax

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

Beispielaufruf:

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

5.2.1.1.33 GetDocumentLength

Diese Methode gibt die Länge eines JSON-Dokuments im DOM-Speicher zurück.

Syntax

```
METHOD GetDocumentLength: UDINT
```

Beispielaufruf:

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

5.2.1.1.34 GetDocumentRoot

Diese Methode liefert den Root-Knoten eines JSON-Dokuments im DOM-Speicher.

Syntax

```
METHOD GetDocumentRoot : SJsonValue
```

Beispielaufruf:

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

5.2.1.1.35 GetDouble

Diese Methode liefert den Value eines Properties vom Datentyp LREAL.

Syntax

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

5.2.1.1.36 GetFileTime

Diese Methode liefert den Value eines Properties vom Datentyp DCTIME.

Syntax

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

Beispielaufruf:

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

5.2.1.1.37 GetHexBinary

Diese Methode dekodiert den HexBinary-Inhalt eines Properties und schreibt diesen an eine bestimmte Speicheradresse, z. B. in eine Datenstruktur.

Syntax

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

Beispielaufruf:

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

5.2.1.1.38 GetInt

Diese Methode liefert den Value eines Properties vom Datentyp DINT.

Syntax

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

5.2.1.1.39 GetInt64

Diese Methode liefert den Value eines Properties vom Datentyp LINT.

Syntax

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

5.2.1.1.40 GetJson

Diese Methode liefert den Value eines Properties als Datentyp STRING(255) zurück, wenn dieser selbst wieder ein JSON-Dokument ist. Bei längeren Zeichenfolgen wird von der Methode ein NULL-String zurückgegeben. In diesem Fall muss die Methode CopyJson [▶ 40]() verwendet werden.

Syntax

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

Beispielaufruf:

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

5.2.1.1.41 GetJsonLength

Diese Methode liefert die Länge eines Properties, wenn dieses ein JSON-Dokument ist.

Syntax

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

Beispielaufruf:

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

5.2.1.1.42 GetMaxDecimalPlaces

Diese Methode liefert die aktuelle Einstellung für MaxDecimalPlaces. Dies beeinflusst die Anzahl der Dezimalstellen bei Fließkommazahlen.

Syntax

```
METHOD GetMaxDecimalPlaces : DINT
```

Beispielaufruf:

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

5.2.1.1.43 GetMemberName

Diese Methode liefert den Namen eines JSON-Property-Members an der Position des aktuellen Iterators, z. B. während der Iteration durch die Kindelemente eines JSON-Properties mit den Methoden MemberBegin(), MemberEnd() und NextMember().

Syntax

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

Beispielaufruf:

```
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

Diese Methode liefert den Value eines JSON-Property-Members an der Position des aktuellen Iterators, z.B. während der Iteration durch die Kindelemente eines JSON-Properties mit den Methoden MemberBegin(), MemberEnd() und NextMember().

Syntax

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

Beispielaufruf:

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

5.2.1.1.45 GetString

Diese Methode liefert den Value eines Properties vom Datentyp STRING(255). Bei längeren Zeichenfolgen wird von der Methode ein NULL-String zurückgegeben. In diesem Fall muss die Methode CopyString [▶ 41]() verwendet werden.

Syntax

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

5.2.1.1.46 GetStringLength

Diese Methode liefert die Länge eines Properties, wenn dessen Value ein String ist.

Syntax

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

Beispielaufruf:

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

5.2.1.1.47 GetType

Diese Methode liefert den Typ eines Property-Values. Der Rückgabewert kann hierbei einen der Werte des Enums EJsonType annehmen.

Syntax

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

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

Beispielaufruf:

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

5.2.1.1.48 GetUint

Diese Methode liefert den Value eines Properties vom Datentyp UDINT.

Syntax

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

5.2.1.1.49 GetUint64

Diese Methode liefert den Value eines Properties vom Datentyp ULINT.

Syntax

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

5.2.1.1.50 HasMember

Diese Methode prüft, ob ein bestimmtes Property im DOM-Speicher vorhanden ist. Wenn das Property vorhanden ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

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

Beispielaufruf:

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

5.2.1.1.51 IsArray

Diese Methode prüft, ob es sich bei einem gegebenen Property um ein Array handelt. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

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

5.2.1.1.52 IsBase64

Diese Methode prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp Base64 handelt. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

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

5.2.1.1.53 IsBool

Diese Methode prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp BOOL handelt. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

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

5.2.1.1.54 IsDouble

Diese Methode prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp Double (SPS: LREAL) handelt. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

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

5.2.1.1.55 IsFalse

Diese Methode prüft, ob der Value eines gegebenen Properties FALSE ist. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

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

5.2.1.1.56 IsHexBinary

Diese Methode prüft, ob der Value eines Properties ein HexBinary-Format hat. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

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

Beispielaufruf:

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

5.2.1.1.57 IsInt

Diese Methode prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp Integer (SPS: DINT) handelt. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

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

5.2.1.1.58 IsInt64

Diese Methode prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp LINT handelt. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

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

5.2.1.1.59 IsISO8601TimeFormat

Diese Methode prüft, ob es sich bei dem Value eines gegebenen Properties um ein Zeitformat laut ISO8601 handelt. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

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

5.2.1.1.60 IsNull

Diese Methode prüft, ob es sich bei dem Value eines gegebenen Properties um NULL handelt. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

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

5.2.1.1.61 IsNumber

Diese Methode prüft, ob es sich bei dem Value eines gegebenen Properties um einen numerischen Wert handelt. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

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

5.2.1.1.62 IsObject

Diese Methode prüft, ob es sich bei dem gegebenen Property um ein weiteres JSON-Objekt handelt. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

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

5.2.1.1.63 IsString

Diese Methode prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp STRING handelt. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

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

5.2.1.1.64 IsTrue

Diese Methode prüft, ob der Wert eines gegebenen Properties TRUE ist. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

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

5.2.1.1.65 IsUint

Diese Methode prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp UDINT handelt. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

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

5.2.1.1.66 IsUint64

Diese Methode prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp ULINT handelt. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

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

5.2.1.1.67 LoadDocumentFromFile

Diese Methode lädt ein JSON-Dokument aus einer Datei.

Eine steigende Flanke am Eingangsparameter bExec triggert hierbei den Ladevorgang. Der asynchrone Prozess ist beendet, sobald die Referenz bExec von der Methode aus zurück auf FALSE gesetzt wird. Mit Beendigung des Prozesses zeigt der Rückgabewert der Methode für einen Aufruf lang an, ob das Laden der Datei erfolgreich (TRUE) oder fehlgeschlagen (FALSE) war.

Syntax

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

Beispielaufruf:

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

5.2.1.1.68 MemberBegin

Diese Methode liefert das erste Kindelemente unterhalb eines JSON-Properties und kann zusammen mit den Methoden MemberEnd() und NextMember() zur Iteration durch ein JSON-Property verwendet werden.

Syntax

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

Beispielaufruf:

```
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

Diese Methode liefert das letzte Kindelement unterhalb eines JSON-Properties und kann zusammen mit den Methoden MemberBegin() und NextMember() zur Iteration durch ein JSON-Property verwendet werden.

Syntax

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

Beispielaufruf:

```
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

Diese Methode erzeugt ein neues, leeres JSON-Dokument im DOM-Speicher.

Syntax

```
METHOD NewDocument : SJsonValue
```

Beispielaufruf:

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

5.2.1.1.71 NextArray

5.2.1.1.72 ParseDocument

Diese Methode lädt ein JSON-Objekt zur weiteren Verarbeitung in den DOM-Speicher. Das JSON-Objekt liegt hierbei als String vor und wird als Eingang an die Methode übergeben. Eine Referenz zum JSON-Dokument im DOM-Speicher wird an den Aufrufer zurückgegeben.

Syntax

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

Beispielaufruf:

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

5.2.1.1.73 PushbackBase64Value

Diese Methode hängt einen Base64-Wert an das Ende eines Arrays an. Als Eingabeparameter kann z. B. eine Struktur adressiert werden. Die entsprechende Base64-Kodierung erfolgt durch die Methode.

Syntax

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

Beispielaufruf:

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

5.2.1.1.74 PushbackBoolValue

Diese Methode hängt einen Wert vom Datentyp BOOL an das Ende eines Arrays an.

Syntax

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

Beispielaufruf:

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

5.2.1.1.75 PushbackDateTimeValue

Diese Methode hängt einen Wert vom Datentyp DATE_AND_TIME an das Ende eines Arrays an.

Syntax

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

Beispielaufruf:

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

5.2.1.1.76 PushbackDcTimeValue

Diese Methode hängt einen Wert vom Datentyp DCTIME an das Ende eines Arrays an.

Syntax

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

Beispielaufruf:

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

5.2.1.1.77 PushbackDoubleValue

Diese Methode hängt einen Wert vom Datentyp Double an das Ende eines Arrays an.

Syntax

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

Beispielaufruf:

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

5.2.1.1.78 PushbackFileTimeValue

Diese Methode hängt einen Wert vom Datentyp FILETIME an das Ende eines Arrays an.

Syntax

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

Beispielaufruf:

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

5.2.1.1.79 PushbackHexBinaryValue

Diese Methode hängt einen HexBinary-kodierten Wert an das Ende eines Arrays an. Die Kodierung in das HexBinary-Format wird durch die Methode durchgeführt. Als Quelle kann z. B. eine Datenstruktur verwendet werden.

Syntax

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

Beispielaufruf:

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

5.2.1.1.80 PushbackInt64Value

Diese Methode hängt einen Wert vom Datentyp Int64 an das Ende eines Arrays an.

Syntax

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

Beispielaufruf:

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

5.2.1.1.81 PushbackIntValue

Diese Methode hängt einen Wert vom Datentyp INT an das Ende eines Arrays an.

Syntax

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

Beispielaufruf:

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

5.2.1.1.82 PushbackJsonValue

Diese Methode fügt ein JSON-Dokument zum Ende eines Arrays hinzu.

Syntax

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

Beispielaufruf:

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

5.2.1.1.83 PushbackNullValue

Diese Methode hängt einen NULL-Wert an das Ende eines Arrays an.

Syntax

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

Beispielaufruf:

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

5.2.1.1.84 PushbackStringValue

Diese Methode hängt einen Wert vom Datentyp DCTIME an das Ende eines Arrays an.

Syntax

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

Beispielaufruf:

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

5.2.1.1.85 PushbackUint64Value

Diese Methode hängt einen Wert vom Datentyp UInt64 an das Ende eines Arrays an.

Syntax

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

Beispielaufruf:

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

5.2.1.1.86 PushbackUintValue

Diese Methode hängt einen Wert vom Datentyp UInt an das Ende eines Arrays an.

Syntax

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

Beispielaufruf:

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

5.2.1.1.87 RemoveAllMembers

Diese Methode entfernt alle Kindelemente von einem gegebenen Property.

Syntax

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

Beispielaufruf:

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

5.2.1.1.88 RemoveArray

Diese Methode löscht den Wert des aktuellen Array-Iterators.

Syntax

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

Beispielaufruf:

Gegeben sei das folgende JSON Dokument, das in den DOM-Speicher geladen wird:

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

Die erste Array-Position soll gelöscht werden. Zunächst wird das JSON-Dokument iterativ nach dem Property „array“ durchsucht, anschließend wird das erste Element des Arrays durch Aufruf der Methode RemoveArray()entfernt.

```
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

Diese Methode löscht das Property an dem aktuellen Iterator.

Syntax

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

Beispielaufruf:

Gegeben sei das folgende JSON Dokument, das in den DOM-Speicher geladen wird:

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

Das Property „array“ soll gelöscht werden. Zunächst wird das JSON-Dokument iterativ nach dem Property „array“ durchsucht, anschließend wird es entfernt.

```
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

Diese Methode entfernt ein Kindelement von einem gegebenen Property. Das Element wird hierbei über dessen Namen referenziert.

Syntax

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

Beispielaufruf:

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

5.2.1.1.91 SaveDocumentToFile

Diese Methode speichert ein JSON-Dokument in einer Datei.

Eine steigende Flanke am Eingangsparameter bExec triggert hierbei den Speichervorgang. Der asynchrone Prozess ist beendet, sobald die Referenz bExec von der Methode aus zurück auf FALSE gesetzt wird. Mit Beendigung des Prozesses zeigt der Rückgabewert der Methode für einen Aufruf lang an, ob das Speichern der Datei erfolgreich (TRUE) oder fehlgeschlagen (FALSE) war.

Syntax

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

Beispielaufruf:

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

5.2.1.1.92 SetArray

Diese Methode setzt den Value eines Properties auf den Typ „Array“. Neue Werte können nun mit den Pushback-Methoden zum Array hinzugefügt werden.

Syntax

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

Beispielaufruf:

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

5.2.1.1.93 SetBase64

Diese Methode setzt den Value eines Properties auf einen Base64-kodierten Wert. Als Quelle kann z. B. eine Datenstruktur verwendet werden. Die Kodierung in das Base64-Format erfolgt innerhalb der Methode.

Syntax

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

Beispielaufruf:

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

5.2.1.1.94 SetBool

Diese Methode setzt den Value eines Properties auf einen Wert vom Datentyp BOOL.

Syntax

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

Beispielaufruf:

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

5.2.1.1.95 SetDateTime

Diese Methode setzt den Value eines Properties auf einen Wert vom Datentyp DATE_AND_TIME.

Syntax

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

Beispielaufruf:

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

5.2.1.1.96 SetDcTime

Diese Methode setzt den Value eines Properties auf einen Wert vom Datentyp DCTIME.

Syntax

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

Beispielaufruf:

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

5.2.1.1.97 SetDouble

Diese Methode setzt den Value eines Properties auf einen Wert vom Datentyp Double.

Syntax

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

Beispielaufruf:

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

5.2.1.1.98 SetFileTime

Diese Methode setzt den Value eines Properties auf einen Wert vom Datentyp FILETIME.

Syntax

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

Beispielaufruf:

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

5.2.1.1.99 SetHexBinary

Diese Methode setzt den Value eines Properties auf einen HexBinary-kodierten Wert. Als Quelle kann z. B. eine Datenstruktur verwendet werden. Die Kodierung in das HexBinary-Format erfolgt innerhalb der Methode.

Syntax

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

Beispielaufruf:

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

5.2.1.1.100 SetInt

Diese Methode setzt den Value eines Properties auf einen Wert vom Datentyp INT.

Syntax

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

Beispielaufruf:

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

5.2.1.1.101 SetInt64

Diese Methode setzt den Value eines Properties auf einen Wert vom Datentyp Int64.

Syntax

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

Beispielaufruf:

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

5.2.1.1.102 SetJson

Diese Methode fügt in den Value eines Properties ein weiteres JSON-Dokument ein.

Syntax

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

Beispielaufruf:

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

5.2.1.1.103 SetMaxDecimalPlaces

Diese Methode setzt die aktuelle Einstellung für MaxDecimalPlaces. Dies stellt die maximal zu verwendende Anzahl an Dezimalstellen bei Fließkommazahlen ein.

Syntax

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

Beispielaufruf:

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

5.2.1.1.104 SetNull

Diese Methode setzt den Value eines Properties auf den Wert NULL.

Syntax

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

Beispielaufruf:

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

5.2.1.1.105 SetObject

Diese Methode setzt den Value eines Properties auf den Typ „Object“. Dies ermöglicht eine Verschachtelung von JSON-Dokumenten.

Syntax

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

Beispielaufruf:

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

5.2.1.1.106 SetString

Diese Methode setzt den Value eines Properties auf einen Wert vom Datentyp STRING.

Syntax

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

Beispielaufruf:

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

5.2.1.1.107 SetUint

Diese Methode setzt den Value eines Properties auf einen Wert vom Datentyp UInt.

Syntax

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

Beispielaufruf:

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

5.2.1.1.108 SetUint64

Diese Methode setzt den Value eines Properties auf einen Wert vom Datentyp UInt64.

Syntax

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

Beispielaufruf:

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

5.2.1.2 FB_JsonDynDomParser

Dieser Funktionsblock ist von demselben internen Funktionsbaustein abgeleitet wie der FB_JsonDomParser [▶ 33] und bietet somit dasselbe Interface.

Die beiden abgeleiteten Funktionsblöcke unterscheiden sich nur in ihrer internen Speicherverwaltung. Der FB_JsonDynDomParser ist optimiert für JSON-Dokumente, an denen viele Änderungen vorgenommen werden und das JSON-Dokument nicht zwischendurch freigegeben wird. Er gibt den allokierten Speicher nach Ausführung einer Aktion, z.B. bei den Methoden SetObject() oder SetJson(), wieder frei. Durch diese Flexibilität ergibt sich ein größerer Overhead, sodass der FB_JsonDomParser [▶ 33] eine bessere Performance ermöglicht.

Strings im UTF-8-Format

Die hier verwendeten Variablen vom Typ STRING nutzen das UTF-8-Format. Diese STRING-Formatierung ist üblich bei IoT/MQTT-Kommunikation sowie JSON-Dokumenten.

Um Sonderzeichen und Texte verschiedenster Sprachen empfangen zu können, wird der Zeichensatz in den Bibliotheken Tc3_IotBase und Tc3_JsonXml nicht auf den typischen Zeichensatz vom Datentyp STRING beschränkt. Stattdessen wird der Unicode-Zeichensatz als UTF-8-Format in Verbindung mit dem Datentyp STRING verwendet.

Bei Verwendung des ASCII-Zeichensatzes besteht kein Unterschied zwischen der typischen Formatierung in einem STRING und der UTF-8-Formatierung eines STRING.

Weitere Informationen zum UTF-8-STRING-Format sowie vorhandenen Anzeige- und Konvertierungsmöglichkeiten finden Sie in der Dokumentation der SPS-Bibliothek Tc2_Utilities.

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4024.7	x86, x64, ARM	Tc3_JsonXml 3.3.8.0

5.2.1.3 FB_JsonSaxReader

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4022	x86, x64, ARM	Tc3_JsonXml

5.2.1.3.1 DecodeBase64

Diese Methode konvertiert einen Base64-formatierten String in Binärdaten. Wenn die Konvertierung erfolgreich war, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

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

Beispielaufruf:

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

5.2.1.3.2 DecodeDateTime

Diese Methode ermöglicht es, aus einem genormten ISO8601-Zeitformat (z. B. YYYY-MM-DDThh:mm:ss) eine SPS-Variable vom Datentyp DATE_AND_TIME bzw. DT zu erzeugen. DT entspricht hierbei der Anzahl an Sekunden ab dem Datum 01.01.1970. Wenn die Konvertierung erfolgreich war, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

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

Beispielaufruf:

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

5.2.1.3.3 DecodeDcTime

Diese Methode ermöglicht es, aus einem genormten ISO8601 Zeitformat (z. B. YYYY-MM-DDThh:mm:ss) eine SPS-Variable vom Datentyp DCTIME zu erzeugen. DCTIME entspricht der Anzahl an Nanosekunden ab dem Datum 01.01.2000. Wenn die Konvertierung erfolgreich war, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

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

Beispielaufruf:

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

5.2.1.3.4 DecodeFileTime

Diese Methode ermöglicht es, aus einem genormten ISO8601-Zeitformat (z. B. YYYY-MM-DDThh:mm:ss) eine SPS-Variable vom Datentyp FILETIME zu erzeugen. FILETIME entspricht der Anzahl an Nanosekunden ab dem Datum 01.01.1601 – gemessen in 100 Nanosekunden. Wenn die Konvertierung erfolgreich war, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

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

Beispielaufruf:

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

5.2.1.3.5 DecodeHexBinary

Diese Methode konvertiert einen String, der Hexadezimalwerte enthält, in Binärdaten um. Wenn die Konvertierung erfolgreich war, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

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

Beispielaufruf:

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

5.2.1.3.6 IsBase64

Diese Methode prüft, ob der übergebene String dem Base64-Format entspricht. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

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

Beispielaufruf:

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

5.2.1.3.7 IsHexBinary

Diese Methode prüft, ob der übergebene String aus Hexadezimalwerten besteht. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

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

Beispielaufruf:

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

5.2.1.3.8 IsISO8601TimeFormat

Diese Methode prüft, ob der übergebene String dem genormten ISO8601-Zeitformat entspricht. Wenn dies der Fall ist, gibt die Methode TRUE zurück, ansonsten FALSE.

Syntax

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

Beispielaufruf:

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

5.2.1.3.9 Parse

Durch diese Methode wird der SAX-Reader-Parsing-Vorgang gestartet. Als Eingangsparameter werden das zu parsende JSON-Objekt und eine Referenz auf einen Funktionsbaustein übergeben, der vom Interface ITcJsonSaxHandler abgeleitet wurde. Dieser Funktionsbaustein wird dann für die Callback-Methoden des SAX Readers verwendet.

Syntax

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

5.2.1.3.10 ParseValues

Durch diese Methode wird der SAX-Reader-Parsing-Vorgang gestartet. Als Eingangsparameter werden das zu parsende JSON-Objekt und eine Referenz auf einen Funktionsbaustein übergeben, der vom Interface ITcJsonSaxValues abgeleitet wurde. Dieser Funktionsbaustein wird dann für die Callback-Methoden des SAX Readers verwendet. Die Besonderheit bei dieser Methode ist, dass bei den Callback-Methoden ausschließlich Values berücksichtigt werden, d. h. es gibt keine OnKey()- oder OnStartObject()-Callbacks.

Syntax

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

5.2.1.4 FB_JsonSaxWriter

Strings im UTF-8-Format

Die hier verwendeten Variablen vom Typ STRING nutzen das UTF-8-Format. Diese STRING-Formatierung ist üblich bei IoT/MQTT-Kommunikation sowie JSON-Dokumenten.

Um Sonderzeichen und Texte verschiedenster Sprachen empfangen zu können, wird der Zeichensatz in den Bibliotheken Tc3_IotBase und Tc3_JsonXml nicht auf den typischen Zeichensatz vom Datentyp STRING beschränkt. Stattdessen wird der Unicode-Zeichensatz als UTF-8-Format in Verbindung mit dem Datentyp STRING verwendet.

Bei Verwendung des ASCII-Zeichensatzes besteht kein Unterschied zwischen der typischen Formatierung in einem STRING und der UTF-8-Formatierung eines STRING.

Weitere Informationen zum UTF-8-STRING-Format sowie vorhandenen Anzeige- und Konvertierungsmöglichkeiten finden Sie in der Dokumentation der SPS-Bibliothek Tc2_Utilities.

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4022	x86, x64, ARM	Tc3_JsonXml

5.2.1.4.1 AddBase64

Diese Methode fügt einen Wert vom Datentyp Base64 zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 69] ein entsprechendes Property erstellt.

Syntax

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

5.2.1.4.2 AddBool

Diese Methode fügt einen Wert vom Datentyp BOOL zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 69] ein entsprechendes Property erstellt.

Syntax

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

Beispielaufruf:

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

5.2.1.4.3 AddDateTime

Diese Methode fügt einen Wert vom Datentyp DATE_AND_TIME zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 69] ein entsprechendes Property erstellt.

Syntax

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

Beispielaufruf:

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

5.2.1.4.4 AddDcTime

Diese Methode fügt einen Wert vom Datentyp DCTIME zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 69] ein entsprechendes Property erstellt.

Syntax

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

Beispielaufruf:

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

5.2.1.4.5 AddDint

Diese Methode fügt einen Wert vom Datentyp DINT zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 69] ein entsprechendes Property erstellt.

Syntax

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

Beispielaufruf:

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

5.2.1.4.6 AddFileTime

Diese Methode fügt einen Wert vom Datentyp FILETIME zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 69] ein entsprechendes Property erstellt.

Syntax

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

Beispielaufruf:

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

5.2.1.4.7 AddHexBinary

Diese Methode fügt einen HexBinary-Wert zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 69] ein entsprechendes Property erstellt.

Syntax

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

Beispielaufruf:

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

5.2.1.4.8 AddKey

Diese Methode fügt einen neuen Property-Key an der aktuellen Position des SAX Writers hinzu. Üblicherweise wird anschließend der Value des neuen Properties gesetzt. Dies kann zum Beispiel über eine der folgenden Methoden erfolgen: AddBase64 [▶ 67], AddBool [▶ 67], AddDateTime [▶ 68], AddDcTime [▶ 68], AddDint [▶ 68], AddFileTime [▶ 68], AddHexBinary [▶ 69], AddLint [▶ 71], AddLreal [▶ 72], AddNull [▶ 72], AddRawArray [▶ 72], AddRawObject [▶ 72], AddReal [▶ 73], AddString [▶ 73], AddUdint [▶ 73], AddUlint [▶ 73].

Syntax

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

Beispielaufruf:

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

5.2.1.4.9 AddKeyBool

Diese Methode legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp BOOL an.

Syntax

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

Beispielaufruf:

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

5.2.1.4.10 AddKeyDateTime

Diese Methode legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp DATE_AND_TIME an.

Syntax

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

Beispielaufruf:

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

5.2.1.4.11 AddKeyDcTime

Diese Methode legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp DCTIME an.

Syntax

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

Beispielaufruf:

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

5.2.1.4.12 AddKeyFileTime

Diese Methode legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp FILETIME an.

Syntax

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

Beispielaufruf:

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

5.2.1.4.13 AddKeyLreal

Diese Methode legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp LREAL an.

Syntax

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

Beispielaufruf:

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

5.2.1.4.14 AddKeyNull

Diese Methode legt einen neuen Property-Key an und initialisiert dessen Wert mit null.

Syntax

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

Beispielaufruf:

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

5.2.1.4.15 AddKeyNumber

Diese Methode legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp DINT an.

Syntax

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

Beispielaufruf:

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

5.2.1.4.16 AddKeyString

Diese Methode legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp STRING an.

Syntax

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

Beispielaufruf:

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

5.2.1.4.17 AddLint

Diese Methode fügt einen Wert vom Datentyp LINT zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 69] ein entsprechendes Property erstellt.

Syntax

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

Beispielaufruf:

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

5.2.1.4.18 AddLreal

Diese Methode fügt einen Wert vom Datentyp LREAL zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 69] ein entsprechendes Property erstellt.

Syntax

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

Beispielaufruf:

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

5.2.1.4.19 AddNull

Diese Methode fügt den Wert null zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 69] ein entsprechendes Property erstellt.

Syntax

```
METHOD AddNull
```

Beispielaufruf:

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

5.2.1.4.20 AddRawArray

Diese Methode fügt ein gültiges JSON-Array zu einem gegebenen Property als Value hinzu. Das hinzuzufügende Array muss ein gültiges JSON-Format aufweisen und darf nur dann hinzugefügt werden, wenn der SAX Writer sich an einer entsprechend gültigen Position befindet, also z. B. direkt nach einem vorhergehenden AddKey() [▶ 69], StartArray() [▶ 75] oder als erster Aufruf nach einem ResetDocument() [▶ 75].

Syntax

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

Beispielaufruf:

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

5.2.1.4.21 AddRawObject

Diese Methode fügt ein gültiges JSON-Objekt zu einem gegebenen Property als Value hinzu. Das hinzuzufügende Objekt muss ein gültiges JSON-Format aufweisen und darf nur dann hinzugefügt werden, wenn der SAX Writer sich an einer entsprechend gültigen Position befindet, also z. B. direkt nach einem vorhergehenden AddKey() [▶ 69], StartArray() [▶ 75] oder als erster Aufruf nach einem ResetDocument() [▶ 75].

Syntax

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

Beispielaufruf:

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

5.2.1.4.22 AddReal

Diese Methode fügt einen Wert vom Datentyp REAL zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 69] ein entsprechendes Property erstellt.

Syntax

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

Beispielaufruf:

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

5.2.1.4.23 AddString

Diese Methode fügt einen Wert vom Datentyp STRING zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 69] ein entsprechendes Property erstellt.

Syntax

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

Beispielaufruf:

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

5.2.1.4.24 AddUdint

Diese Methode fügt einen Wert vom Datentyp UDINT zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 69] ein entsprechendes Property erstellt.

Syntax

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

Beispielaufruf:

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

5.2.1.4.25 AddUlint

Diese Methode fügt einen Wert vom Datentyp ULINT zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 69] ein entsprechendes Property erstellt.

Syntax

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

Beispielaufruf:

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

5.2.1.4.26 CopyDocument

Diese Methode kopiert den Inhalt des aktuell mit dem SAX Writer erstellten JSON-Objekts in eine Zielvariable vom Datentyp STRING, welche eine beliebige Länge haben kann. Als Rückgabewert liefert die Methode die Länge des Strings (inklusive Nullterminierung). Falls der Zielpuffer zu klein ist, wird dieser durch eine Nullterminierung geleert und als Länge 0 zurückgegeben.

Syntax

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

Beispielaufruf:

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

5.2.1.4.27 EndArray

Diese Methode erzeugt den Abschluss eines angefangenen JSON-Arrays („eckige schließende Klammer“) und fügt sie an der aktuellen Position des SAX Writers ein.

Syntax

```
METHOD EndArray : HRESULT
```

Beispielaufruf:

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

5.2.1.4.28 EndObject

Diese Methode erzeugt den Abschluss eines angefangenen JSON-Objekts („geschweifte schließende Klammer“) und fügt sie an der aktuellen Position des SAX Writers ein.

Syntax

```
METHOD EndObject : HRESULT
```

Beispielaufruf:

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

5.2.1.4.29 GetDocument

Diese Methode liefert den Inhalt des aktuell mit dem SAX Writer erstellten JSON-Objekts und gibt diesen als Datentyp STRING(255) zurück.

Die maximale Größe des von der Methode zurückgelieferten Strings ist 255 Zeichen. Bei längeren Zeichenfolgen wird von der Methode ein NULL-String zurückgegeben. In diesem Fall muss die Methode CopyDocument [▶ 74]() verwendet werden.

Syntax

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

Beispielaufruf:

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

5.2.1.4.30 GetDocumentLength

Diese Methode liefert die Länge des aktuell mit dem SAX Writer erstellten JSON-Objekts und gibt diese als Datentyp UDINT zurück.

Syntax

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

Beispielaufruf:

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

5.2.1.4.31 GetMaxDecimalPlaces

Syntax

```
METHOD GetMaxDecimalPlaces : DINT
```

5.2.1.4.32 ResetDocument

Diese Methode setzt das aktuell mit dem SAX Writer erstellte JSON-Objekt zurück.

Syntax

```
METHOD ResetDocument : HRESULT
```

Beispielaufruf:

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

5.2.1.4.33 SetMaxDecimalPlaces

Syntax

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

5.2.1.4.34 StartArray

Diese Methode erzeugt den Anfang eines neuen JSON-Arrays („eckige öffnende Klammer“) und fügt sie an der aktuellen Position des SAX Writers ein.

Syntax

```
METHOD StartArray : HRESULT
```

Beispielaufruf:

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

5.2.1.4.35 StartObject

Diese Methode erzeugt den Beginn eines neuen JSON-Objekts („geschweifte öffnende Klammer“) und fügt sie an der aktuellen Position des SAX Writers ein.

Syntax

```
METHOD StartObject : HRESULT
```

Beispielaufruf:

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

5.2.1.5 FB_JsonSaxPrettyWriter

Dieser Funktionsbaustein hat einen Unterschied zum FB_JsonSaxWriter [▶ 67]: Es werden für eine bessere Lesbarkeit eines JSON-Dokuments passende Whitespaces hinzugefügt.

Strings im UTF-8-Format

Die hier verwendeten Variablen vom Typ STRING nutzen das UTF-8-Format. Diese STRING-Formatierung ist üblich bei IoT/MQTT-Kommunikation sowie JSON-Dokumenten.

Um Sonderzeichen und Texte verschiedenster Sprachen empfangen zu können, wird der Zeichensatz in den Bibliotheken Tc3_IotBase und Tc3_JsonXml nicht auf den typischen Zeichensatz vom Datentyp STRING beschränkt. Stattdessen wird der Unicode-Zeichensatz als UTF-8-Format in Verbindung mit dem Datentyp STRING verwendet.

Bei Verwendung des ASCII-Zeichensatzes besteht kein Unterschied zwischen der typischen Formatierung in einem STRING und der UTF-8-Formatierung eines STRING.

Weitere Informationen zum UTF-8-STRING-Format sowie vorhandenen Anzeige- und Konvertierungsmöglichkeiten finden Sie in der Dokumentation der SPS-Bibliothek Tc2_Utilities.

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4022	x86, x64, ARM	Tc3_JsonXml

5.2.1.5.1 AddBase64

Diese Methode fügt einen Wert vom Datentyp Base64 zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 78] ein entsprechendes Property erstellt.

Syntax

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

5.2.1.5.2 AddBool

Diese Methode fügt einen Wert vom Datentyp BOOL zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 78] ein entsprechendes Property erstellt.

Syntax

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

Beispielaufruf:

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

5.2.1.5.3 AddDateTime

Diese Methode fügt einen Wert vom Datentyp DATE_AND_TIME zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 78] ein entsprechendes Property erstellt.

Syntax

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

Beispielaufruf:

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

5.2.1.5.4 AddDcTime

Diese Methode fügt einen Wert vom Datentyp DCTIME zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 78] ein entsprechendes Property erstellt.

Syntax

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

Beispielaufruf:

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

5.2.1.5.5 AddDint

Diese Methode fügt einen Wert vom Datentyp DINT zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 78] ein entsprechendes Property erstellt.

Syntax

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

Beispielaufruf:

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

5.2.1.5.6 AddFileTime

Diese Methode fügt einen Wert vom Datentyp FILETIME zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 78] ein entsprechendes Property erstellt.

Syntax

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

Beispielaufruf:

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

5.2.1.5.7 AddHexBinary

Diese Methode fügt einen HexBinary-Wert zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 78] ein entsprechendes Property erstellt.

Syntax

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

Beispielaufruf:

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

5.2.1.5.8 AddKey

Diese Methode fügt einen neuen Property-Key an der aktuellen Position des SAX Writers hinzu. Üblicherweise wird anschließend der Value des neuen Properties gesetzt. Dies kann zum Beispiel über eine der folgenden Methoden erfolgen: AddBase64 [▶ 76], AddBool [▶ 76], AddDateTime [▶ 77], AddDcTime [▶ 77], AddDint [▶ 77], AddFileTime [▶ 77], AddHexBinary [▶ 78], AddLint [▶ 80], AddLreal [▶ 81], AddNull [▶ 81], AddRawArray [▶ 81], AddRawObject [▶ 81], AddReal [▶ 82], AddString [▶ 82], AddUdint [▶ 82], AddUlint [▶ 82].

Syntax

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

Beispielaufruf:

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

5.2.1.5.9 AddKeyBool

Diese Methode legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp BOOL an.

Syntax

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

Beispielaufruf:

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

5.2.1.5.10 AddKeyDateTime

Diese Methode legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp DATE_AND_TIME an.

Syntax

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

Beispielaufruf:

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

5.2.1.5.11 AddKeyDcTime

Diese Methode legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp DCTIME an.

Syntax

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

Beispielaufruf:

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

5.2.1.5.12 AddKeyFileTime

Diese Methode legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp FILETIME an.

Syntax

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

Beispielaufruf:

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

5.2.1.5.13 AddKeyLreal

Diese Methode legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp LREAL an.

Syntax

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

Beispielaufruf:

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

5.2.1.5.14 AddKeyNull

Diese Methode legt einen neuen Property-Key an und initialisiert dessen Wert mit null.

Syntax

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

Beispielaufruf:

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

5.2.1.5.15 AddKeyNumber

Diese Methode legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp DINT an.

Syntax

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

Beispielaufruf:

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

5.2.1.5.16 AddKeyString

Diese Methode legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp STRING an.

Syntax

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

Beispielaufruf:

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

5.2.1.5.17 AddLint

Diese Methode fügt einen Wert vom Datentyp LINT zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 78] ein entsprechendes Property erstellt.

Syntax

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

Beispielaufruf:

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

5.2.1.5.18 AddLreal

Diese Methode fügt einen Wert vom Datentyp LREAL zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 78] ein entsprechendes Property erstellt.

Syntax

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

Beispielaufruf:

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

5.2.1.5.19 AddNull

Diese Methode fügt den Wert null zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 78] ein entsprechendes Property erstellt.

Syntax

```
METHOD AddNull
```

Beispielaufruf:

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

5.2.1.5.20 AddRawArray

Diese Methode fügt ein gültiges JSON-Array zu einem gegebenen Property als Value hinzu. Das hinzuzufügende Array muss ein gültiges JSON-Format aufweisen und darf nur dann hinzugefügt werden, wenn der SAX Writer sich an einer entsprechend gültigen Position befindet, also z. B. direkt nach einem vorhergehenden AddKey() [▶ 78], StartArray() [▶ 85] oder als erster Aufruf nach einem ResetDocument() [▶ 84].

Syntax

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

Beispielaufruf:

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

5.2.1.5.21 AddRawObject

Diese Methode fügt ein gültiges JSON-Objekt zu einem gegebenen Property als Value hinzu. Das hinzuzufügende Objekt muss ein gültiges JSON-Format aufweisen und darf nur dann hinzugefügt werden, wenn der SAX Writer sich an einer entsprechend gültigen Position befindet, also z. B. direkt nach einem vorhergehenden AddKey() [▶ 78], StartArray() [▶ 85] oder als erster Aufruf nach einem ResetDocument() [▶ 84].

Syntax

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

Beispielaufruf:

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

5.2.1.5.22 AddReal

Diese Methode fügt einen Wert vom Datentyp REAL zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 78] ein entsprechendes Property erstellt.

Syntax

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

Beispielaufruf:

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

5.2.1.5.23 AddString

Diese Methode fügt einen Wert vom Datentyp STRING zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 78] ein entsprechendes Property erstellt.

Syntax

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

Beispielaufruf:

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

5.2.1.5.24 AddUdint

Diese Methode fügt einen Wert vom Datentyp UDINT zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 78] ein entsprechendes Property erstellt.

Syntax

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

Beispielaufruf:

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

5.2.1.5.25 AddUlint

Diese Methode fügt einen Wert vom Datentyp ULINT zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 78] ein entsprechendes Property erstellt.

Syntax

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

Beispielaufruf:

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

5.2.1.5.26 CopyDocument

Diese Methode kopiert den Inhalt des aktuell mit dem SAX Writer erstellten JSON-Objekts in eine Zielvariable vom Datentyp STRING, welche eine beliebige Länge haben kann. Als Rückgabewert liefert die Methode die Länge des Strings (inklusive Nullterminierung). Falls der Zielpuffer zu klein ist, wird dieser durch eine Nullterminierung geleert und als Länge 0 zurückgegeben.

Syntax

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

Beispielaufruf:

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

5.2.1.5.27 EndArray

Diese Methode erzeugt den Abschluss eines angefangenen JSON-Arrays („eckige schließende Klammer“) und fügt sie an der aktuellen Position des SAX Writers ein.

Syntax

```
METHOD EndArray : HRESULT
```

Beispielaufruf:

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

5.2.1.5.28 EndObject

Diese Methode erzeugt den Abschluss eines angefangenen JSON-Objekts („geschweifte schließende Klammer“) und fügt sie an der aktuellen Position des SAX Writers ein.

Syntax

```
METHOD EndObject : HRESULT
```

Beispielaufruf:

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

5.2.1.5.29 GetDocument

Diese Methode liefert den Inhalt des aktuell mit dem SAX Writer erstellten JSON-Objekts und gibt diesen als Datentyp STRING(255) zurück.

Die maximale Größe des von der Methode zurückgelieferten Strings ist 255 Zeichen. Bei längeren Zeichenfolgen wird von der Methode ein NULL-String zurückgegeben. In diesem Fall muss die Methode CopyDocument [▶ 83]() verwendet werden.

Syntax

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

Beispielaufruf:

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

5.2.1.5.30 GetDocumentLength

Diese Methode liefert die Länge des aktuell mit dem SAX Writer erstellten JSON-Objekts und gibt diese als Datentyp UDINT zurück.

Syntax

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

Beispielaufruf:

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

5.2.1.5.31 GetMaxDecimalPlaces

Syntax

```
METHOD GetMaxDecimalPlaces : DINT
```

5.2.1.5.32 ResetDocument

Diese Methode setzt das aktuell mit dem SAX Writer erstellte JSON-Objekt zurück.

Syntax

```
METHOD ResetDocument : HRESULT
```

Beispielaufruf:

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

5.2.1.5.33 SetFormatOptions

Diese Methode wird verwendet, um die allgemeinen Formatierungsoptionen für eine Instanz des FB_JsonSaxPrettyWriter [▶ 76] anzupassen. Zusätzlich zu der Standardkonfiguration gibt es noch die Möglichkeit, dass innerhalb eines Arrays keine Zeilenumbrüche erzeugt werden. Das Array wird dann trotz der eingefügten Einrückungen in einer Zeile ausgegeben.

Syntax

```
METHOD SetFormatOptions : HRESULT
VAR_INPUT
  options : EJsonPrettyFormatOptions;
END_VAR
TYPE EJsonPrettyFormatOptions:
(
```

```
  eFormatDefault:=0,
  eFormatSingleLineArray:=1
) DINT;
```

Beispielaufruf:

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

5.2.1.5.34 SetIndent

Diese Methode wird verwendet, um die Einrückung benutzerdefiniert anzupassen.

Syntax

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

Beispielaufruf:

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

5.2.1.5.35 SetMaxDecimalPlaces

Syntax

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

5.2.1.5.36 StartArray

Diese Methode erzeugt den Anfang eines neuen JSON-Arrays („eckige öffnende Klammer“) und fügt sie an der aktuellen Position des SAX Writers ein.

Syntax

```
METHOD StartArray : HRESULT
```

Beispielaufruf:

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

5.2.1.5.37 StartObject

Diese Methode erzeugt den Beginn eines neuen JSON-Objekts („geschweifte öffnende Klammer“) und fügt sie an der aktuellen Position des SAX Writers ein.

Syntax

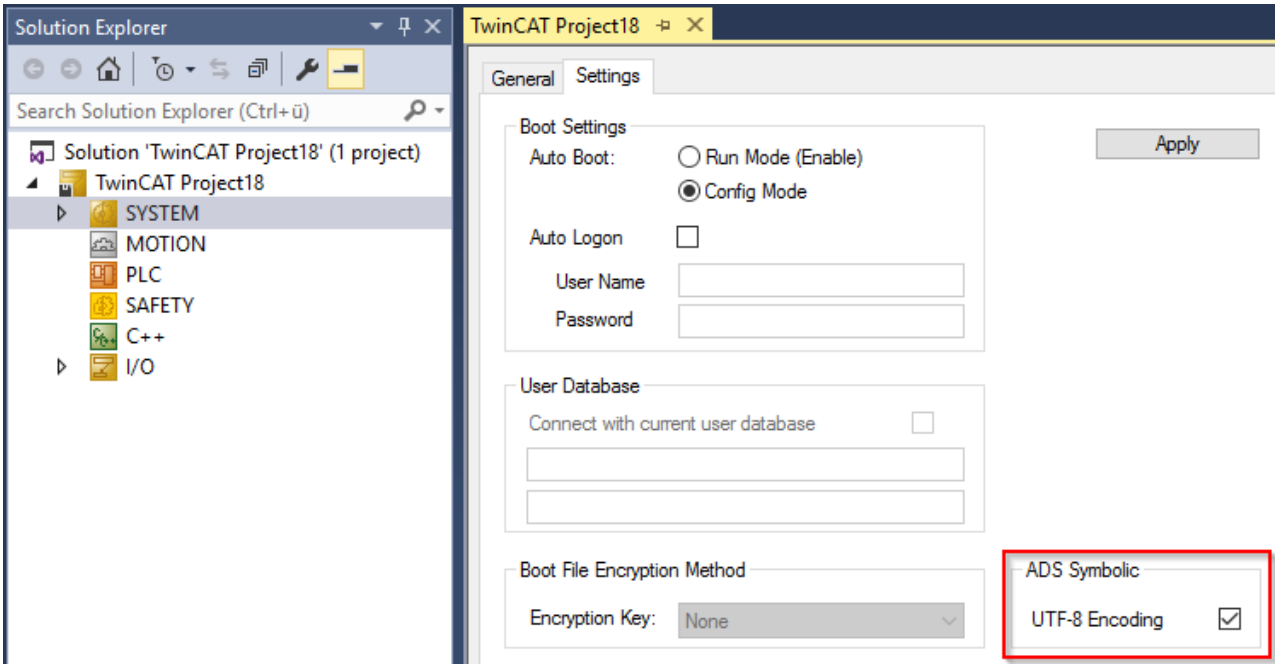
```
METHOD StartObject : HRESULT
```

Beispielaufruf:

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

5.2.1.6 FB_JsonReadWriteDataType

Zur Verwendung von UTF-8-Zeichen, z. B. bei der automatischen Generierung von Metadaten über den Funktionsbaustein FB_JsonReadWriteDataType [▶ 86], muss im TwinCAT-Projekt das Auswahlkästchen zur Unterstützung von UTF-8 in der Symbolik aktiviert sein. Klicken Sie dazu im Projektbaum doppelt auf **SYSTEM**, öffnen Sie die Registerkarte **Settings** und aktivieren Sie das entsprechende Auswahlkästchen.



Strings im UTF-8-Format

Die hier verwendeten Variablen vom Typ STRING nutzen das UTF-8-Format. Diese STRING-Formatierung ist üblich bei IoT/MQTT-Kommunikation sowie JSON-Dokumenten.

Um Sonderzeichen und Texte verschiedenster Sprachen empfangen zu können, wird der Zeichensatz in den Bibliotheken Tc3_IotBase und Tc3_JsonXml nicht auf den typischen Zeichensatz vom Datentyp STRING beschränkt. Stattdessen wird der Unicode-Zeichensatz als UTF-8-Format in Verbindung mit dem Datentyp STRING verwendet.

Bei Verwendung des ASCII-Zeichensatzes besteht kein Unterschied zwischen der typischen Formatierung in einem STRING und der UTF-8-Formatierung eines STRING.

Weitere Informationen zum UTF-8-STRING-Format sowie vorhandenen Anzeige- und Konvertierungsmöglichkeiten finden Sie in der Dokumentation der SPS-Bibliothek Tc2_Utilities.

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4022	x86, x64, ARM	Tc3_JsonXml

5.2.1.6.1 AddJsonKeyPropertiesFromSymbol

Mit dieser Methode können Metadaten über SPS-Attribute in die JSON-Repräsentation einer SPS-Datenstruktur auf einem FB_JsonSaxWriter [▶ 67]-Objekt hinzugefügt werden. Als Eingangsparameter erhält die Methode die Instanz des FB_JsonSaxWriter-Funktionsbausteins, den gewünschten Namen des JSON-Properties, das die Metadaten enthalten soll, den Datentypnamen der Struktur und eine String-Variable sProperties, die eine durch einen Querbalken getrennte Liste der zu extrahierenden SPS-Attribute enthält.

Syntax

```
METHOD AddJsonValueFromSymbol : BOOL
VAR_IN_OUT
  fbWriter    : FB_JsonSaxWriter;
```

```
END_VAR
VAR_IN_OUT CONSTANT
  sKey        : STRING;
  sDatatype   : STRING;
  sProperties : STRING;
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

Die SPS-Attribute können in folgender Form an den Strukturelementen spezifiziert werden:

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

Ein vollständiges Beispiel zur Verwendung dieser Methode finden Sie im Abschnitt Tc3JsonXmlSampleJsonDataType.

Beispielaufruf:

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

5.2.1.6.2 AddJsonKeyValueFromSymbol

Diese Methode erzeugt die JSON-Repräsentation einer SPS-Datenstruktur auf einem FB_JsonSaxWriter [▶ 67]-Objekt. Als Eingangsparameter erhält die Methode die Instanz des FB_JsonSaxWriter-Funktionsbausteins, den Datentypnamen der Struktur sowie die Adresse und Größe der Quellstrukturinstanz. Als Resultat beinhaltet die FB_JsonSaxWriter-Instanz eine gültige JSON-Repräsentation der Struktur. Im Unterschied zur Methode AddJsonValueFromSymbol() [▶ 87] werden die Elemente der Quellstruktur hierbei in ein JSON-Unterobjekt verschachtelt, dessen Name über den Eingangs-/Ausgangsparameter sKey spezifiziert werden kann.

Syntax

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

Ein vollständiges Beispiel zur Verwendung dieser Methode finden Sie im Abschnitt Tc3JsonXmlSampleJsonDataType.

Beispielaufruf:

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

5.2.1.6.3 AddJsonValueFromSymbol

Diese Methode erzeugt die JSON-Repräsentation einer SPS-Datenstruktur auf einem FB_JsonSaxWriter [▶ 67]-Objekt. Als Eingangsparameter erhält die Methode die Instanz des FB_JsonSaxWriter-Funktionsbausteins, den Datentypnamen der Struktur sowie die Adresse und Größe der Quellstrukturinstanz. Als Resultat beinhaltet die FB_JsonSaxWriter-Instanz eine gültige JSON-Repräsentation der Struktur.

Syntax

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

Ein vollständiges Beispiel zur Verwendung dieser Methode finden Sie im Abschnitt Tc3JsonXmlSampleJsonDataType.

Beispielaufruf:

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

5.2.1.6.4 CopyJsonStringFromSymbol

Diese Methode erzeugt die JSON-Repräsentanz eines Symbols und kopiert diese in eine Variable vom Datentyp STRING, welche eine beliebige Länge haben kann. Als Rückgabewert liefert die Methode die Länge des Strings (inklusive Nullterminierung). Falls der Zielpuffer zu klein ist, wird dieser durch eine Nullterminierung geleert und als Länge 0 zurückgegeben.

Syntax

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

Beispielaufruf:

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

5.2.1.6.5 CopyJsonStringFromSymbolProperties

Diese Methode erzeugt eine entsprechende JSON-Repräsentation von SPS-Attributen an einem Symbol. Im Unterschied zur Methode AddJsonKeyPropertiesFromSymbol [▶ 86] wird das Resultat nicht in eine Instanz vom Funktionsbaustein FB_JsonSaxWriter geschrieben, sondern in eine String-Variable. Als Eingangsparameter erhält die Methode den Datentypnamen des Symbols und eine String-Variable, welche eine durch Querbalken getrennte Liste der zu extrahierenden SPS-Attribute darstellt.

Die Methode kopiert diese JSON-Repräsentation in eine Variable vom Datentyp STRING, welche eine beliebige Länge haben kann. Als Rückgabewert liefert die Methode die Länge des Strings (inklusive Nullterminierung). Falls der Zielpuffer zu klein ist, wird dieser durch eine Nullterminierung geleert und als Länge 0 zurückgegeben.

Syntax

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

Beispielaufruf:

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

5.2.1.6.6 CopySymbolNameByAddress

Diese Methode liefert den vollständigen (ADS-)Symbolnamen eines übergebenen Symbols. Als Rückgabewert liefert die Methode die Größe des Strings (inklusive Nullterminierung). Falls der Zielpuffer zu klein ist, wird dieser durch eine Nullterminierung geleert und als Länge 0 zurückgegeben.

Syntax

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

Beispielaufruf:

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

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4024.20	x86, x64, ARM	Tc3_JsonXml 3.3.15.0

5.2.1.6.7 GetDataTypeNameByAddress

Diese Methode liefert den Datentypnamen eines übergebenen Symbols.

Syntax

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

Beispielaufruf:

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

5.2.1.6.8 GetJsonFromSymbol

Diese Methode erzeugt die entsprechende JSON-Repräsentation eines Symbols. Im Unterschied zur Methode AddJsonValueFromSymbol() [▶ 87] wird das Resultat nicht in eine Instanz vom Funktionsbaustein FB_JsonSaxWriter geschrieben, sondern in eine String-Variable. Als Eingangsparameter erhält die Methode den Datentypnamen des Symbols sowie die Adresse und Größe des Quellsymbols, z. B. einer Strukturinstanz. Als weitere Eingangsparameter werden die Adresse und Größe des Ziel-Buffers übergeben, der nach dem Aufruf die JSON-Repräsentation des Symbols enthält.

Syntax

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

Eingangsparameter nJson

Der Eingangsparameter nJson enthält beim Aufruf der Methode die Größe des Ziel-Buffers und nach Abschluss des Methodenaufrufs die Größe des tatsächlich geschriebenen JSON-Objekts im Ziel-Buffer.

Beispielaufruf:

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

5.2.1.6.9 GetJsonStringFromSymbol

Diese Methode erzeugt die entsprechende JSON-Repräsentation eines Symbols. Im Unterschied zur Methode AddJsonValueFromSymbol() [▶ 87] wird das Resultat nicht in eine Instanz vom Funktionsbaustein FB_JsonSaxWriter geschrieben, sondern in eine String-Variable. Als Eingangsparameter erhält die Methode den Datentypnamen des Symbols sowie die Adresse und die Größe des Quellsymbols, z. B. einer Strukturinstanz.

Die maximale Größe des von der Methode zurückgelieferten Strings ist 255 Zeichen. Bei längeren Zeichenfolgen wird von der Methode ein NULL-String zurückgegeben. In diesem Fall muss die Methode CopyJsonStringFromSymbol [▶ 88]() verwendet werden.

Syntax

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

Beispielaufruf:

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

5.2.1.6.10 GetJsonStringFromSymbolProperties

Diese Methode erzeugt eine entsprechende JSON-Repräsentation von SPS-Attributen an einem Symbol. Im Unterschied zur Methode AddJsonKeyPropertiesFromSymbol [▶ 86] wird das Resultat nicht in eine Instanz vom Funktionsbaustein FB_JsonSaxWriter geschrieben, sondern in eine String-Variable. Als Eingangsparameter erhält die Methode den Datentypnamen des Symbols und eine String-Variable, welche eine durch Querbalken getrennte Liste der zu extrahierenden SPS-Attribute darstellt. Das Resultat wird direkt als Rückgabewert der Methode zurückgeliefert.

Die maximale Größe des von der Methode zurückgelieferten Strings ist 255 Zeichen. Bei längeren Zeichenfolgen wird von der Methode ein NULL-String zurückgegeben. In diesem Fall muss die Methode CopyJsonStringFromSymbolProperties [▶ 88]() verwendet werden.

Syntax

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

Beispielaufruf:

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

5.2.1.6.11 GetSizeJsonStringFromSymbol

Diese Methode liest die Größe der JSON-Repräsentanz eines Symbols aus. Der Wert wird dabei mit Nullterminierung angegeben.

Syntax

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

Beispielaufruf:

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

5.2.1.6.12 GetSizeJsonStringFromSymbolProperties

Diese Methode liest die Größe der JSON-Repräsentanz von SPS-Attributen an einem Symbol aus. Der Wert wird dabei mit Nullterminierung angegeben.

Syntax

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

Beispielaufruf:

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

5.2.1.6.13 GetSymbolNameByAddress

Diese Methode liefert den vollständigen (ADS-)Symbolnamen eines übergebenen Symbols.

Die maximale Größe des von der Methode zurückgelieferten Strings ist 255 Zeichen. Bei längeren Zeichenfolgen wird von der Methode ein Null-String zurückgegeben. In diesem Fall muss die Methode CopySymbolNameByAddress() [▶ 89] verwendet werden.

Syntax

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

Beispielaufruf:

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

5.2.1.6.14 SetSymbolFromJson

Diese Methode extrahiert einen String, der eine gültige JSON-Nachricht beinhaltet, und versucht die Inhalte des JSON-Objekts in eine äquivalente Datenstruktur zu speichern. Als Eingangsparameter erhält die Methode den String mit dem JSON-Objekt, den Datentypnamen der Zielstruktur sowie die Adresse und Größe der Zielstrukturinstanz.

Syntax

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

Beispielaufruf:

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

5.2.1.7 FB_XmlDomParser

Strings im UTF-8-Format

Die hier verwendeten Variablen vom Typ STRING nutzen das UTF-8-Format. Diese STRING-Formatierung ist üblich bei IoT/MQTT-Kommunikation sowie JSON-Dokumenten.

Um Sonderzeichen und Texte verschiedenster Sprachen empfangen zu können, wird der Zeichensatz in den Bibliotheken Tc3_IotBase und Tc3_JsonXml nicht auf den typischen Zeichensatz vom Datentyp STRING beschränkt. Stattdessen wird der Unicode-Zeichensatz als UTF-8-Format in Verbindung mit dem Datentyp STRING verwendet.

Bei Verwendung des ASCII-Zeichensatzes besteht kein Unterschied zwischen der typischen Formatierung in einem STRING und der UTF-8-Formatierung eines STRING.

Weitere Informationen zum UTF-8-STRING-Format sowie vorhandenen Anzeige- und Konvertierungsmöglichkeiten finden Sie in der Dokumentation der SPS-Bibliothek Tc2_Utilities.

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4022	x86, x64, ARM	Tc3_JsonXml

5.2.1.7.1 AppendAttribute

Diese Methode fügt ein neues Attribut zu einem existierenden Knoten hinzu. Der Name und der Wert des neuen Attributs sowie der existierende XML-Knoten werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf das neu hinzugefügte Attribut.

Syntax

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

Beispielaufruf:

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

5.2.1.7.2 AppendAttributeAsBool

Diese Methode fügt ein neues Attribut zu einem existierenden Knoten hinzu. Der Wert des Attributs ist hierbei vom Datentyp Boolean. Der Name und der Wert des neuen Attributs sowie der existierende XML-Knoten werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf das neu hinzugefügte Attribut.

Syntax

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

Beispielaufruf:

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

5.2.1.7.3 AppendAttributeAsDouble

Diese Methode fügt ein neues Attribut zu einem existierenden Knoten hinzu. Der Wert des Attributs ist hierbei vom Datentyp Double. Der Name und der Wert des neuen Attributs sowie der existierende XML-Knoten werden der Methode als Eingangsparameter übergeben. Die Methode returniert eine Referenz auf das neu hinzugefügte Attribut.

Syntax

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

```
VAR_INPUT
  value : LREAL;
END_VAR
```

Beispielaufruf:

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

5.2.1.7.4 AppendAttributeAsFloat

Diese Methode fügt ein neues Attribut zu einem existierenden Knoten hinzu. Der Wert des Attributs ist hierbei vom Datentyp Float. Der Name und der Wert des neuen Attributs sowie der existierende XML-Knoten werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf das neu hinzugefügte Attribut.

Syntax

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

Beispielaufruf:

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

5.2.1.7.5 AppendAttributeAsInt

Diese Methode fügt ein neues Attribut zu einem existierenden Knoten hinzu. Der Wert des Attributs ist hierbei vom Datentyp Integer. Der Name und der Wert des neuen Attributs sowie der existierende XML-Knoten werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf das neu hinzugefügte Attribut.

Syntax

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

Beispielaufruf:

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

5.2.1.7.6 AppendAttributeAsLint

Diese Methode fügt ein neues Attribut zu einem existierenden Knoten hinzu. Der Wert des Attributs ist hierbei vom Datentyp Integer64. Der Name und der Wert des neuen Attributs sowie der existierende XML-Knoten werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf das neu hinzugefügte Attribut.

Syntax

```
METHOD AppendAttributeAsLint : SXmlAttribute
VAR_INPUT
  n : SXmlNode;
END_VAR
```

```
VAR_IN_OUT CONSTANT
  name : STRING;
END_VAR
VAR_INPUT
  value : LINT;
END_VAR
```

Beispielaufruf:

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

5.2.1.7.7 AppendAttributeAsUint

Diese Methode fügt ein neues Attribut zu einem existierenden Knoten hinzu. Der Wert des Attributs ist hierbei vom Datentyp Unsigned Integer. Der Name und der Wert des neuen Attributs sowie der existierende XML-Knoten werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf das neu hinzugefügte Attribut.

Syntax

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

Beispielaufruf:

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

5.2.1.7.8 AppendAttributeAsUlint

Diese Methode fügt ein neues Attribut zu einem existierenden Knoten hinzu. Der Wert des Attributs ist hierbei vom Datentyp Unsigned Integer64. Der Name und der Wert des neuen Attributs sowie der existierende XML-Knoten werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf das neu hinzugefügte Attribut.

Syntax

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

Beispielaufruf:

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

5.2.1.7.9 AppendAttributeCopy

Diese Methode fügt ein neues Attribut zu einem existierenden Knoten hinzu. Der Name und der Wert des neuen Attributs werden hierbei von einem existierenden Attribut übernommen bzw. kopiert. Das existierende Attribut wird der Methode als Eingangsparameter übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.10 AppendChild

Diese Methode fügt einen neuen Knoten unterhalb eines existierenden Knotens ein. Der Wert des neuen Knotens ist hierbei vom Datentyp STRING. Der Name und der Wert des neuen Knotens sowie eine Referenz auf den existierenden Knoten werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf den neu hinzugefügten Knoten. Der Eingangsparameter cdata gibt an, ob der Wert des Knotens in einem CDATA-Block gekapselt werden soll, sodass bestimmte Sonderzeichen wie z. B. "<" und ">" als Werte erlaubt sind.

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
```

Beispielaufruf:

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

5.2.1.7.11 AppendChildAsBool

Diese Methode fügt einen neuen Knoten unterhalb eines existierenden Knotens ein. Der Wert des neuen Knotens ist hierbei vom Datentyp Boolean. Der Name und der Wert des neuen Knotens sowie eine Referenz auf den existierenden Knoten werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf den neu hinzugefügten Knoten.

Syntax

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

Beispielaufruf:

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

5.2.1.7.12 AppendChildAsDouble

Diese Methode fügt einen neuen Knoten unterhalb eines existierenden Knotens ein. Der Wert des neuen Knotens ist hierbei vom Datentyp Double. Der Name und der Wert des neuen Knotens sowie eine Referenz auf den existierenden Knoten werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf den neu hinzugefügten Knoten.

Syntax

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

Beispielaufruf:

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

5.2.1.7.13 AppendChildAsFloat

Diese Methode fügt einen neuen Knoten unterhalb eines existierenden Knotens ein. Der Wert des neuen Knotens ist hierbei vom Datentyp Float. Der Name und der Wert des neuen Knotens sowie eine Referenz auf den existierenden Knoten werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf den neu hinzugefügten Knoten.

Syntax

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

Beispielaufruf:

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

5.2.1.7.14 AppendChildAsInt

Diese Methode fügt einen neuen Knoten unterhalb eines existierenden Knotens ein. Der Wert des neuen Knotens ist hierbei vom Datentyp Integer. Der Name und der Wert des neuen Knotens sowie eine Referenz auf den existierenden Knoten werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf den neu hinzugefügten Knoten.

Syntax

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

Beispielaufruf:

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

5.2.1.7.15 AppendChildAsLint

Diese Methode fügt einen neuen Knoten unterhalb eines existierenden Knotens ein. Der Wert des neuen Knotens ist hierbei vom Datentyp Integer64. Der Name und der Wert des neuen Knotens sowie eine Referenz auf den existierenden Knoten werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf den neu hinzugefügten Knoten.

Syntax

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

Beispielaufruf:

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

5.2.1.7.16 AppendChildAsUint

Diese Methode fügt einen neuen Knoten unterhalb eines existierenden Knotens ein. Der Wert des neuen Knotens ist hierbei vom Datentyp Unsigned Integer. Der Name und der Wert des neuen Knotens sowie eine Referenz auf den existierenden Knoten werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf den neu hinzugefügten Knoten.

Syntax

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

Beispielaufruf:

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

5.2.1.7.17 AppendChildAsUlint

Diese Methode fügt einen neuen Knoten unterhalb eines existierenden Knotens ein. Der Wert des neuen Knotens ist hierbei vom Datentyp Unsigned Integer64. Der Name und der Wert des neuen Knotens sowie eine Referenz auf den existierenden Knoten werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf den neu hinzugefügten Knoten.

Syntax

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

Beispielaufruf:

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

5.2.1.7.18 AppendCopy

Diese Methode fügt einen neuen Knoten unterhalb eines existierenden Knotens ein. Der Name und der Wert des neuen Knotens werden von einem existierenden Knoten übernommen bzw. kopiert. Die Referenzen auf die existierenden Knoten werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf den neu hinzugefügten Knoten.

Syntax

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

Beispielaufruf:

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

5.2.1.7.19 AppendNode

Diese Methode fügt einen neuen Knoten zu einem existierenden Knoten hinzu. Der existierende Knoten sowie der Name des neuen Knotens werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf den neu angefügten Knoten.

Syntax

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

Beispielaufruf:

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

5.2.1.7.20 Attribute

Mit dieser Methode kann das Attribut eines gegebenen XML-Knotens ausgelesen werden. Der XML-Knoten und der Name des Attributs werden der Methode als Eingangsparameter übergeben. Nach Aufruf der Methode müssen weitere Methoden aufgerufen werden, um z. B. den Wert des Attributs auszulesen, z. B. AttributeAsInt().

Syntax

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

Beispielaufruf:

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

5.2.1.7.21 AttributeAsBool

Diese Methode liefert den Wert eines Attributs als Datentyp Boolean. Das Attribut wird der Methode als Eingangsparameter übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.22 AttributeAsDouble

Diese Methode liefert den Wert eines Attributs als Datentyp Double. Das Attribut wird der Methode als Eingangsparameter übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.23 AttributeAsFloat

Diese Methode liefert den Wert eines Attributs als Datentyp Float. Das Attribut wird der Methode als Eingangsparameter übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.24 AttributeAsInt

Diese Methode liefert den Wert eines Attributs als Datentyp Integer. Das Attribut wird der Methode als Eingangsparameter übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.25 AttributeAsLint

Diese Methode liefert den Wert eines Attributs als Datentyp Integer64. Das Attribut wird der Methode als Eingangsparameter übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.26 AttributeAsUint

Diese Methode liefert den Wert eines Attributs als Datentyp Unsigned Integer. Das Attribut wird der Methode als Eingangsparameter übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.27 AttributeAsUlint

Diese Methode liefert den Wert eines Attributs als Datentyp Unsigned Integer64. Das Attribut wird der Methode als Eingangsparameter übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.28 AttributeBegin

Diese Methode liefert einen Iterator über alle Attribute eines XML-Knotens. Der XML-Knoten wird der Methode als Eingangsparameter übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.29 AttributeFromIterator

Diese Methode konvertiert die aktuelle Position eines Iterators in ein XML-Attribut-Objekt. Der Iterator wird der Methode als Eingangsparameter übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.30 AttributeName

Diese Methode liefert den Namen eines gegebenen Attributs. Das Attribut wird der Methode als Eingangsparameter übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.31 Attributes

Diese Methode dient zur Navigation durch den DOM und liefert einen Iterator auf alle gefundenen Attribute an einem XML-Knoten zurück. Der Iterator kann anschließend zur weiteren Navigation durch die gefundenen Elemente verwendet werden. Der Knoten und eine Referenz auf den Iterator werden der Methode als Eingangsparameter übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.32 AttributeText

Diese Methode liefert den Text eines gegebenen Attributs. Das Attribut wird der Methode als Eingangsparameter übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.33 Begin

Diese Methode liefert einen Iterator über alle Kindelemente eines XML-Knotens, immer beginnend ab dem ersten Kindelement. Der XML-Knoten wird der Methode als Eingangsparameter übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.34 BeginByName

Diese Methode liefert einen Iterator über alle Kindelemente eines XML-Knotens, beginnend ab einem bestimmten Element. Der XML-Knoten wird der Methode als Eingangsparameter übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.35 Child

Diese Methode dient zur Navigation durch den DOM und liefert eine Referenz auf das (erste) Kindelement des aktuellen Knotens zurück. Der Startknoten wird der Methode als Eingangsparameter übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.36 ChildByAttribute

Diese Methode dient zur Navigation durch den DOM und liefert eine Referenz auf ein Kindelement im XML-Dokument zurück. Der Startknoten sowie der Name und der Wert des Attributs werden der Methode als Eingangsparameter übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.37 ChildByAttributeAndName

Diese Methode dient zur Navigation durch den DOM und liefert eine Referenz auf ein Kindelement im XML-Dokument zurück. Der Startknoten, der Name und der Wert des Attributs sowie der Name des Kindelements werden der Methode als Eingangsparameter übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.38 ChildByName

Diese Methode dient zur Navigation durch den DOM und liefert eine Referenz auf ein Kindelement im XML-Dokument zurück. Der Startknoten und der Name des zurückzuliefernden Elements werden der Methode als Eingangsparameter übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.39 Children

Diese Methode dient zur Navigation durch den DOM und liefert einen Iterator auf mehrere gefundene Kindelemente im XML-Dokument zurück. Der Iterator kann dann zur weiteren Navigation durch die gefundenen Elemente verwendet werden. Der Startknoten und eine Referenz auf den Iterator werden der Methode als Eingangsparameter übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.40 ChildrenByName

Diese Methode dient zur Navigation durch den DOM und liefert einen Iterator auf mehrere gefundene Kindelemente im XML-Dokument zurück. Der Iterator kann dann zur weiteren Navigation durch die gefundenen Elemente verwendet werden. Der Startknoten sowie der Name der zu suchenden Kindelemente und eine Referenz auf den Iterator werden der Methode als Eingangsparameter übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.41 Compare

Diese Methode überprüft zwei Iteratoren auf Gleichheit.

Syntax

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

Beispielaufruf:

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

5.2.1.7.42 CopyAttributeText

Diese Methode liest den Wert eines XML-Attributs aus und schreibt diesen in eine Variable vom Datentyp String. Das XML-Attribut sowie die Zielvariable und die zu schreibende Länge werden der Methode als Eingangsparameter übergeben. Die tatsächliche Größe wird von der Methode zurückgeliefert.

Syntax

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

Beispielaufruf:

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

5.2.1.7.43 CopyDocument

Diese Methode kopiert den Inhalt des DOM-Speichers in eine Variable vom Datentyp String. Die zu schreibende Länge und die Variable, in die der resultierende String geschrieben werden soll, werden der Methode als Eingangsparameter übergeben. Die tatsächlich geschriebene Länge wird von der Methode zurückgeliefert. Beachten Sie, dass die Größe der String-Variablen mindestens der Größe des XML-Dokuments im DOM entspricht.

Syntax

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

Beispielaufruf:

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

5.2.1.7.44 CopyNodeText

Diese Methode liest den Wert eines XML-Knotens aus und schreibt diesen in eine Variable vom Datentyp String. Der XML-Knoten sowie die Zielvariable und die zu schreibende Länge werden der Methode als Eingangsparameter übergeben. Die tatsächliche Größe wird von der Methode zurückgeliefert.

Syntax

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

Beispielaufruf:

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

5.2.1.7.45 CopyNodeXml

Diese Methode liest die XML-Struktur eines XML Knotens aus und schreibt diese in eine Variable vom Datentyp String. Der XML-Knoten sowie die Zielvariable und die zu schreibende Länge werden der Methode als Eingangsparameter übergeben. Die tatsächliche Größe wird von der Methode zurückgeliefert.

Syntax

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

Beispielaufruf:

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

5.2.1.7.46 FirstNodeByPath

Diese Methode navigiert anhand eines an die Methode übergebenen Pfads durch ein XML-Dokument. Der Pfad und der Startknoten werden der Methode als Eingangsparameter übergeben. Der Pfad wird mit "/" als Separator spezifiziert. Als Rückgabewert liefert die Methode eine Referenz auf den gefundenen XML-Knoten.

Syntax

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

Beispielaufruf:

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

5.2.1.7.47 GetAttributeTextLength

Diese Methode liefert die Länge des Werts eines XML-Attributs. Das XML-Attribut wird der Methode als Eingangsparameter übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.48 GetDocumentLength

Diese Methode liefert die Länge eines XML-Dokuments in Bytes.

Syntax

```
METHOD GetDocumentLength : UDINT
```

Beispielaufruf:

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

5.2.1.7.49 GetDocumentNode

Diese Methode liefert den Root-Knoten eines XML-Dokuments. Dieser ist nicht gleichbedeutend mit dem ersten XML-Knoten im Dokument (verwenden Sie hierfür die Methode GetRootNode()). Die Methode kann auch verwendet werden, um ein leeres XML-Dokument im DOM zu erzeugen.

Syntax

```
METHOD GetDocumentNode : SXmlNode
```

Beispielaufruf:

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

5.2.1.7.50 GetNodeTextLength

Diese Methode liefert die Länge des Werts eines XML-Knotens. Der XML-Knoten wird der Methode als Eingangsparameter übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.51 GetNodeXmlLength

Diese Methode liefert die Länge der XML-Struktur eines XML-Knotens. Der XML-Knoten wird der Methode als Eingangsparameter übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.52 GetRootNode

Diese Methode liefert eine Referenz zum ersten XML-Knoten im XML-Dokument.

Syntax

```
METHOD GetRootNode : SXmlNode
```

Beispielaufruf:

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

5.2.1.7.53 InsertAttributeCopy

Diese Methode fügt ein Attribut zu einem XML-Knoten hinzu und kopiert hierbei den Name und den Wert eines existierenden Attributs. Das Attribut kann hierbei an einer bestimmten Stelle platziert werden. Der XML-Knoten, die Position und eine Referenz auf das existierende Attribut-Objekt werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf das neu hinzugefügte Attribut.

Syntax

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

Beispielaufruf:

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

5.2.1.7.54 InsertAttribute

Diese Methode fügt ein Attribut zu einem XML-Knoten hinzu. Das Attribut kann hierbei an einer bestimmten Stelle platziert werden. Der XML-Knoten, die Position und der Name des neuen Attributs werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf das neu hinzugefügte Attribut. Anschließend kann z. B. über die Methode SetAttribute() ein Wert für das Attribut eingetragen werden.

Syntax

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

Beispielaufruf:

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

5.2.1.7.55 InsertChild

Diese Methode fügt einen Knoten zu einem existierenden XML-Knoten hinzu. Der neue Knoten kann hierbei an einer bestimmten Stelle platziert werden. Der existierende XML-Knoten sowie die Position und der Name des neuen Knotens werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf den neu hinzugefügten Knoten. Anschließend kann z. B. über die Methode SetChild() ein Wert für den Knoten eingetragen werden.

Syntax

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

Beispielaufruf:

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

5.2.1.7.56 InsertCopy

Diese Methode fügt einen neuen Knoten zu einem existierenden XML Knoten hinzu und kopiert hierbei einen existierenden Knoten. Der neue Knoten kann hierbei an einer beliebigen Stelle im existierenden Knoten platziert werden. Der XML-Knoten, die Position und eine Referenz auf das existierende Knoten-Objekt werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf den neu hinzugefügten Knoten.

Syntax

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

Beispielaufruf:

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

5.2.1.7.57 IsEnd

Diese Methode überprüft, ob sich ein gegebener XML-Iterator am Ende der zu durchlaufenden Iteration befindet.

Syntax

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

Beispielaufruf:

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

5.2.1.7.58 LoadDocumentFromFile

Diese Methode lädt ein XML-Dokument aus einer Datei. Der absolute Pfad zu der Datei wird der Methode als Eingangsparameter übergeben.

Eine steigende Flanke am Eingangsparameter bExec triggert hierbei den Ladevorgang. Der asynchrone Prozess ist beendet, sobald die Referenz bExec von der Methode aus zurück auf FALSE gesetzt wird. Mit Beendigung des Prozesses zeigt der Rückgabewert der Methode für einen Aufruf lang an, ob das Laden der Datei erfolgreich (TRUE) oder fehlgeschlagen (FALSE) war.

Syntax

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

Beispielaufruf:

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

5.2.1.7.59 NewDocument

Diese Methode erzeugt ein leeres XML-Dokument im DOM-Speicher.

Syntax

```
METHOD NewDocument : BOOL
```

Beispielaufruf:

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

5.2.1.7.60 Next

Diese Methode setzt einen XML-Iterator auf das nächste zu durchlaufende Objekt.

Syntax

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

Beispielaufruf:

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

5.2.1.7.61 NextAttribute

Diese Methode liefert zu einem gegebenen XML-Attribut das nächste Attribut.

Syntax

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

Beispielaufruf:

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

5.2.1.7.62 NextByName

Diese Methode setzt einen XML-Iterator auf das nächste zu durchlaufende Objekt, das anhand seines Namens identifiziert wird.

Syntax

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

Beispielaufruf:

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

5.2.1.7.63 NextSibling

Diese Methode liefert zu einem gegebenen XML-Knoten den nächsten direkten Knoten auf derselben XML-Ebene.

Syntax

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

Beispielaufruf:

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

5.2.1.7.64 NextSiblingByName

Diese Methode liefert zu einem gegebenen XML-Knoten den nächsten direkten Knoten mit einem bestimmten Namen auf derselben XML-Ebene.

Syntax

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

Beispielaufruf:

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

5.2.1.7.65 Node

Diese Methode wird in Zusammenhang mit einem Iterator verwendet, um durch den DOM zu navigieren. Der Iterator wird der Methode als Eingangsparameter übergeben. Als Rückgabewert liefert die Methode dann den aktuellen XML-Knoten.

Syntax

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

Beispielaufruf:

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

5.2.1.7.66 NodeAsBool

Diese Methode liefert den Text eines XML-Knotens als Datentyp Boolean. Der XML-Knoten wird der Methode als Eingangsparameter übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.67 NodeAsDouble

Diese Methode liefert den Text eines XML-Knotens als Datentyp Double. Der XML-Knoten wird der Methode als Eingangsparameter übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.68 NodeAsFloat

Diese Methode liefert den Text eines XML-Knotens als Datentyp Float. Der XML-Knoten wird der Methode als Eingangsparameter übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.69 NodeAsInt

Diese Methode liefert den Text eines XML-Knotens als Datentyp Integer. Der XML-Knoten wird der Methode als Eingangsparameter übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.70 NodeAsLint

Diese Methode liefert den Text eines XML-Knotens als Datentyp Integer64. Der XML-Knoten wird der Methode als Eingangsparameter übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.71 NodeAsUint

Diese Methode liefert den Text eines XML-Knotens als Datentyp Unsigned Integer. Der XML-Knoten wird der Methode als Eingangsparameter übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.72 NodeAsUlint

Diese Methode liefert den Text eines XML-Knotens als Datentyp Unsigned Integer64. Der XML-Knoten wird der Methode als Eingangsparameter übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.73 NodeName

Diese Methode gibt den Namen eines XML-Knotens zurück. Eine Referenz zum XML-Knoten wird der Methode als Eingangsparameter übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.74 NodeText

Diese Methode liefert den Text eines XML-Knotens. Der XML-Knoten wird der Methode als Eingangsparameter übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.75 ParseDocument

Diese Methode lädt ein XML-Dokument zur weiteren Verarbeitung in den DOM-Speicher. Das XML-Dokument liegt hierbei als String vor und wird der Methode als Eingangsparameter übergeben. Eine Referenz zum XML-Dokument im DOM wird an den Aufrufer zurückgegeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.76 RemoveChild

Diese Methode entfernt einen XML-Kindknoten von einem gegebenen XML-Knoten. Die beiden XML-Knoten werden der Methode als Eingangsparameter übergeben. Die Methode gibt TRUE zurück, wenn der Vorgang erfolgreich war und der XML-Knoten entfernt wurde.

Syntax

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

Beispielaufruf:

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

5.2.1.7.77 RemoveChildByName

Diese Methode entfernt einen XML-Kindknoten von einem gegebenen XML-Knoten. Der zu entfernende Knoten wird über dessen Namen angesprochen. Gibt es mehr als einen Kindknoten, so wird der letzte entfernt. Die Methode gibt TRUE zurück, wenn der Vorgang erfolgreich war und der XML-Knoten entfernt wurde.

Syntax

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

Beispielaufruf:

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

5.2.1.7.78 SaveDocumentToFile

Diese Methode speichert das aktuelle XML-Dokument in einer Datei. Der absolute Pfad zu der Datei wird der Methode als Eingangsparameter übergeben.

Eine steigende Flanke am Eingangsparameter bExec triggert hierbei den Speichervorgang. Der asynchrone Prozess ist beendet, sobald die Referenz bExec von der Methode aus zurück auf FALSE gesetzt wird. Mit Beendigung des Prozesses zeigt der Rückgabewert der Methode für einen Aufruf lang an, ob das Speichern der Datei erfolgreich (TRUE) oder fehlgeschlagen (FALSE) war.

Syntax

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

Beispielaufruf:

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

5.2.1.7.79 SetAttribute

Diese Methode setzt den Wert eines Attributs. Der Wert ist hierbei vom Datentyp String.

Syntax

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

Beispielaufruf:

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

5.2.1.7.80 SetAttributeAsBool

Diese Methode setzt den Wert eines Attributs. Der Wert ist hierbei vom Datentyp Boolean.

Syntax

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

Beispielaufruf:

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

5.2.1.7.81 SetAttributeAsDouble

Diese Methode setzt den Wert eines Attributs. Der Wert ist hierbei vom Datentyp Double.

Syntax

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

Beispielaufruf:

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

5.2.1.7.82 SetAttributeAsFloat

Diese Methode setzt den Wert eines Attributs. Der Wert ist hierbei vom Datentyp Float.

Syntax

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

Beispielaufruf:

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

5.2.1.7.83 SetAttributeAsInt

Diese Methode setzt den Wert eines Attributs. Der Wert ist hierbei vom Datentyp Integer.

Syntax

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

Beispielaufruf:

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

5.2.1.7.84 SetAttributeAsLint

Diese Methode setzt den Wert eines Attributs. Der Wert ist hierbei vom Datentyp Integer64.

Syntax

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

Beispielaufruf:

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

5.2.1.7.85 SetAttributeAsUint

Diese Methode setzt den Wert eines Attributs. Der Wert ist hierbei vom Datentyp Unsigned Integer.

Syntax

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

Beispielaufruf:

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

5.2.1.7.86 SetAttributeAsUlint

Diese Methode setzt den Wert eines Attributs. Der Wert ist hierbei vom Datentyp Unsigned Integer64.

Syntax

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

Beispielaufruf:

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

5.2.1.7.87 SetChild

Diese Methode setzt den Wert eines XML-Knotens. Der Wert wird der Methode als Eingangsparameter vom Datentyp String übergeben. Der Eingangsparameter cdata gibt an, ob der Wert des Knotens in einem CDATA-Block gekapselt werden soll, sodass bestimmte Sonderzeichen wie z. B. "<" und ">" als Werte erlaubt sind.

Syntax

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

Beispielaufruf:

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

5.2.1.7.88 SetChildAsBool

Diese Methode setzt den Wert eines XML-Knotens. Der Wert wird der Methode als Eingangsparameter vom Datentyp Boolean übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.89 SetChildAsDouble

Diese Methode setzt den Wert eines XML-Knotens. Der Wert wird der Methode als Eingangsparameter vom Datentyp Double übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.90 SetChildAsFloat

Diese Methode setzt den Wert eines XML-Knotens. Der Wert wird der Methode als Eingangsparameter vom Datentyp Float übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.91 SetChildAsInt

Diese Methode setzt den Wert eines XML-Knotens. Der Wert wird der Methode als Eingangsparameter vom Datentyp Integer übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.92 SetChildAsLint

Diese Methode setzt den Wert eines XML-Knotens. Der Wert wird der Methode als Eingangsparameter vom Datentyp Integer64 übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.93 SetChildAsUint

Diese Methode setzt den Wert eines XML-Knotens. Der Wert wird der Methode als Eingangsparameter vom Datentyp Unsigned Integer übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.7.94 SetChildAsUlint

Diese Methode setzt den Wert eines XML-Knotens. Der Wert wird der Methode als Eingangsparameter vom Datentyp Unsigned Integer64 übergeben.

Syntax

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

Beispielaufruf:

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

5.2.1.8 FB_JwtEncode

Der Funktionsbaustein ermöglicht die Erzeugung und Signierung eines JSON Web Token (JWT).

Syntax

Definition:

```
FUNCTION_BLOCK FB_JwtEncode
VAR_INPUT
  bExecute       : BOOL;
  sHeaderAlg     : STRING(46);
  sPayload       : STRING(1023);
  sKeyFilePath   : STRING(511);
  tTimeout       : TIME;
  pKey           : PVOID;
  nKeySize       : UDINT;
  nJwtSize       : UDINT;
END_VAR
VAR_IN_OUT CONSTANT
  sJwt           : STRING;
END_VAR
VAR_OUTPUT
  bBusy          : BOOL;
  bError         : BOOL;
  hrErrorCode    : HRESULT;
  initStatus     : HRESULT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
bExecute	BOOL	Steigende Flanke aktiviert die Abarbeitung des Funktionsbausteins.
sHeaderAlg	STRING(46)	Der zu verwendende Algorithmus für den JWT Header, z. B. RS256.
sPayload	STRING(1023	Der zu verwendende Payload des JWT.
sKeyFilePath	STRING(511)	Pfad zum Private Key, welcher für die Signatur des JWT verwendet werden soll.
tTimeout	TIME	ADS Timeout, welcher intern für den Dateizugriff auf den Private Key verwendet wird.
pKey	PVOID	Buffer für den auszulesenden Private Key.
nKeySize	UDINT	Maximalgröße des Buffers.
sJwt	STRING	Enthält nach Abarbeitung des Funktionsbausteins das fertig kodierte und signierte JWT.
nJwtSize	UDINT	Größe des erzeugten JWT inklusive Nullterminierung.

Ausgänge

Name	Typ	Beschreibung
bBusy	BOOL	Ist TRUE, solange die Abarbeitung des Funktionsbausteins noch nicht beendet ist.
bError	BOOL	Wird TRUE, sobald eine Fehlersituation eintritt.
hrErrorCode	HRESULT	Liefert bei einem gesetzten Ausgang `bError` einen Fehlercode. Eine Erläuterung zu den möglichen Fehlercodes befindet sich im Anhang.
initStatus	HRESULT	Liefert im Fall einer fehlgeschlagenen Initialisierung des Funktionsbausteins einen Fehlercode.

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4024.4	x86, x64, ARM	Tc3_JsonXml 3.3.6.0

5.2.2 Schnittstellen

5.2.2.1 ITcJsonSaxHandler

5.2.2.1.1 OnBool

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein Wert vom Datentyp BOOL gefunden wurde. Der Eingangsparameter value enthält hierbei den gefundenen Wert. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnBool : HRESULT
VAR_INPUT
    value : BOOL;
END_VAR
```

5.2.2.1.2 OnDint

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein Wert vom Datentyp DINT gefunden wurde. Der Eingangsparameter value enthält hierbei den gefundenen Wert. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnDint : HRESULT
VAR_INPUT
    value : DINT;
END_VAR
```

5.2.2.1.3 OnEndArray

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers eine eckige schließende Klammer gefunden wurde, was dem JSON-Synonym für ein endendes Array entspricht. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnEndArray : HRESULT
```

5.2.2.1.4 OnEndObject

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers eine geschweifte schließende Klammer gefunden wurde, was dem JSON-Synonym für ein endendes Objekt entspricht. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnEndObject : HRESULT
```

5.2.2.1.5 OnKey

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein Property gefunden wurde. Der Property-Name liegt hierbei am Eingangs-/Ausgangsparameter key an und dessen Länge am Eingangsparameter len. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnKey : HRESULT
VAR_IN_OUT CONSTANT
    key : STRING;
END_VAR
VAR_INPUT
    len : UDINT;
END_VAR
```

5.2.2.1.6 OnLint

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein Wert vom Datentyp LINT gefunden wurde. Der Eingangsparameter value enthält hierbei den gefundenen Wert. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnLint : HRESULT
VAR_INPUT
    value : LINT;
END_VAR
```

5.2.2.1.7 OnLreal

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein Wert vom Datentyp LREAL gefunden wurde. Der Eingangsparameter value enthält hierbei den gefundenen Wert. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnLreal : HRESULT
VAR_INPUT
    value : LREAL;
END_VAR
```

5.2.2.1.8 OnNull

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein NULL-Wert gefunden wurde. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnNull : HRESULT
```

5.2.2.1.9 OnStartArray

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers eine eckige öffnende Klammer gefunden wurde, was dem JSON-Synonym für ein beginnendes Array entspricht. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnStartArray : HRESULT
```

5.2.2.1.10 OnStartObject

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers eine geschweifte öffnende Klammer gefunden wurde, was dem JSON-Synonym für ein beginnendes Objekt entspricht. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnStartObject : HRESULT
```

5.2.2.1.11 OnString

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein Wert vom Datentyp STRING gefunden wurde. Der In/Out-Parameter value enthält hierbei den gefundenen Wert. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnString : HRESULT
VAR_IN_OUT CONSTANT
    value : STRING;
END_VAR
VAR_INPUT
    len : UDINT;
END_VAR
```

5.2.2.1.12 OnUdint

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein Wert vom Datentyp UDINT gefunden wurde. Der Eingangsparameter value enthält hierbei den gefundenen Wert. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnUdint : HRESULT
VAR_INPUT
    value : UDINT;
END_VAR
```

5.2.2.1.13 OnUlint

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein Wert vom Datentyp ULINT gefunden wurde. Der Eingangsparameter value enthält hierbei den gefundenen Wert. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnUlint : HRESULT
VAR_INPUT
    value : ULINT;
END_VAR
```

5.2.2.2 ITcJsonSaxValues

5.2.2.2.1 OnBoolValue

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein Wert vom Datentyp BOOL gefunden wurde. Der Eingangsparameter value enthält hierbei den gefundenen Wert. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnBoolValue : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
    value : BOOL;
END_VAR
```

5.2.2.2.2 OnDintValue

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein Wert vom Datentyp DINT gefunden wurde. Der Eingangsparameter value enthält hierbei den gefundenen Wert. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnDintValue : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
    value : DINT;
END_VAR
```

5.2.2.2.3 OnLintValue

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein Wert vom Datentyp LINT gefunden wurde. Der Eingangsparameter value enthält hierbei den gefundenen Wert. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

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

5.2.2.2.4 OnLrealValue

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein Wert vom Datentyp LREAL gefunden wurde. Der Eingangsparameter value enthält hierbei den gefundenen Wert. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

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

5.2.2.2.5 OnNullValue

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein NULL-Wert gefunden wurde. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

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

5.2.2.2.6 OnStringValue

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein Wert vom Datentyp STRING gefunden wurde. Der Eingangs-/Ausgangsparameter value enthält hierbei den gefundenen Wert. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnStringValue : HRESULT
VAR_IN_OUT CONSTANT
    value : STRING;
END_VAR
VAR_INPUT
    len : UDINT;
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
END_VAR
```

5.2.2.2.7 OnUdintValue

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein Wert vom Datentyp UDINT gefunden wurde. Der Eingangsparameter value enthält hierbei den gefundenen Wert. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

```
METHOD OnUdintValue : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
    value : UDINT;
END_VAR
```

5.2.2.2.8 OnUlintValue

Diese Callback-Methode wird getriggert, wenn an der Position des SAX Readers ein Wert vom Datentyp ULINT gefunden wurde. Der Eingangsparameter value enthält hierbei den gefundenen Wert. Durch Setzen des Rückgabewerts HRESULT auf S_FALSE wird der SAX-Parsing-Vorgang abgebrochen.

Syntax

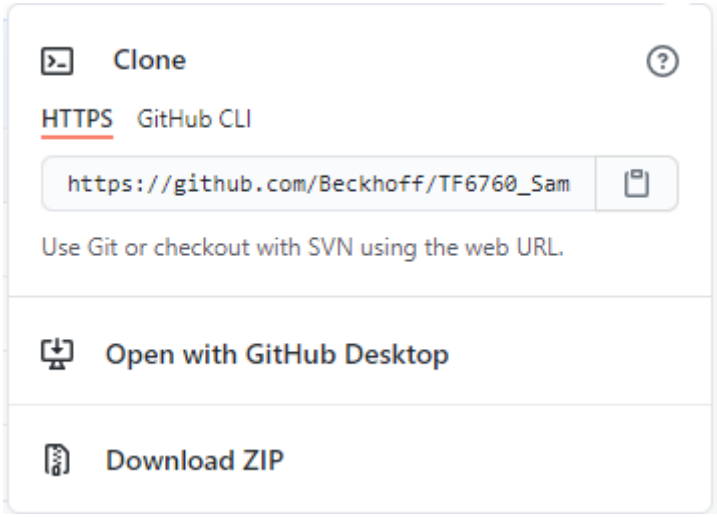
```
METHOD OnUlintValue : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
    value : ULINT;
END_VAR
```

6 Beispiele

6.1 IotHttpSamples

Jedes Beispiel wird als separates SPS-Projekt bereitgestellt. Es sind Beispiele für AWS IoT Core, OpenWeatherMap, Philips Hue, Postman Echo,Telegram Messenger und für die AWS-Service-Konfiguration vorhanden.

Beispielcode und -konfigurationen für dieses Produkt können über das entsprechende Repository auf GitHub bezogen werden: https://github.com/Beckhoff/TF6760_Samples. Sie haben dort die Möglichkeit das Repository zu clonen oder ein ZIP File mit dem Sample herunterzuladen.

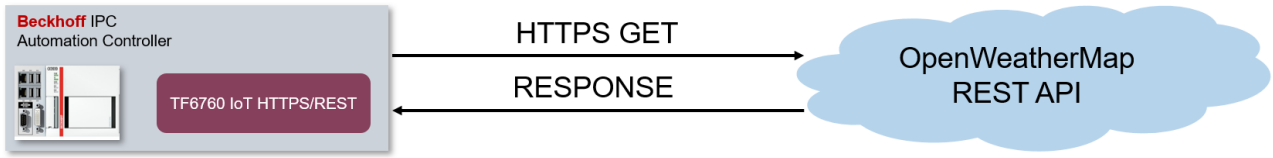


Die SPS-Projekte sind in einen MAIN-Teil und verschiedene Funktionsbausteine aufgeteilt. Die HTTP-befehlsspezifischen Funktionen (GET, POST, PUT) sind in den genannten Funktionsbausteinen gekapselt, um das Programm übersichtlich zu halten.

Der MAIN-Teil jedes Beispiels wird für die Konfiguration der HTTP-Client-Funktionsbausteine verwendet. Wenn die Client-Funktionsbausteine konfiguriert sind, können die verschiedenen HTTP-Befehle aus der MAIN heraus ausgelöst werden. Die instanziierten HTTP-Client-Funktionsbausteine werden dann durch die verschiedenen gekapselten Funktionsbausteine durchgereicht.

6.1.1 OpenWeatherMap

Dieses Beispiel zeigt die Kommunikation mit einer HTTPS/REST-API. HTTPS GET-Anfragen werden gesendet, um Wetterdaten von der REST-API „OpenWeatherMap“ zu erhalten. Die Antwort des Webservice mit den Daten erfolgt in einem JSON-Format.



Um Anfragen an die OpenWeatherMap REST-API senden zu können, muss der Benutzer ein Konto für OpenWeatherMap erstellen, um eine Anwendungs-ID zu erhalten. In Kombination mit einer Standortkennung (z. B. Breitengrad/Längengrad, Stadt-ID, Stadtname) kann die Anwendungs-ID für die Abfrage der Wetterdaten verwendet werden.

Die API-Dokumentation für verschiedene Anwendungsfälle (16-Tage-Vorhersage, aktuelle Daten usw.) finden Sie hier: https://openweathermap.org/api.

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

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

    fbHttpGetOpenWeatherMap         : FB_TestHTTP_Get_openWeatherMap;
END_VAR
```

```
//init client parameters at startup
IF NOT fbHttpClientOpenWeatherMap.bConfigured THEN
    fbHttpClientOpenWeatherMap.nHostPort:= 443;
    fbHttpClientOpenWeatherMap.stTLS.bNoServerCertCheck:= TRUE;
END_IF

IF fbHttpClientOpenWeatherMap.bConfigured THEN
    fbHttpGetOpenWeatherMap(bSend:=bGetOpenWeatherMap, fbClient:=fbHttpClientOpenWeatherMap);
END_IF

fbHttpClientOpenWeatherMap.Execute();
```

6.1.1.1 Get

1. Der Funktionsbaustein wird gestartet, indem die Variable bGetOpenWeatherMap im Hauptprogramm auf TRUE geschrieben wird.
2. Die steigende Flanke wird dann verwendet, um eine GET-Anfrage mit dem Funktionsbaustein IotHttpRequest zu senden.
3. Nachdem die Anfrage abgeschlossen ist, wird die Fehlerbehandlung durchlaufen. Wenn weder der Funktionsbaustein selbst noch der HTTP-Statuscode einen Fehler aufweisen, wird die JSON-Antwort vom Webserver geparst.

```
FUNCTION_BLOCK FB_TestHTTP_Get_openWeatherMap
VAR_INPUT
    bSend               : BOOL;
END_VAR
VAR_IN_OUT
    fbClient            : FB_IotHttpClient;
END_VAR
VAR_OUTPUT
    bBusy               : BOOL;
    bError              : BOOL;
END_VAR
VAR
    fbRequest           : FB_IotHttpRequest;
    fbJson              : FB_JsonDomParser;
    nState              : UDINT;
    RisingEdge          : R_TRIG;

    bGetContentResult   : BOOL;
    sContent            : STRING(511);

    bGetJsonResult      : BOOL;
    jsonDoc             : SJsonValue;
    jsonVal             : SJsonValue;
    sResultValue        : STRING;

    nReqCount           : UDINT;
    nResCount           : UDINT;
    nValidResCount      : UDINT;
    nErrCount           : UDINT;
END_VAR
```

```
RisingEdge(CLK:= bSend);
CASE nState OF
0:
    IF RisingEdge.Q THEN
        IF fbRequest.SendRequest(sUri:= '/data/2.5/weather?id=123456&APPID=123456abcdef',
                                 fbClient:= fbClient,
                                 eRequestType:= ETcIotHttpRequestType.HTTP_Get, 0, 0, 0) THEN
            nState:= 1;
            nReqCount:= nReqCount+1;
            bBusy:= TRUE;
            bError:= FALSE;
        END_IF
    END_IF
```

```
1:
    IF NOT fbRequest.bBusy THEN
        bError:= TRUE;
        IF NOT fbRequest.bError THEN
            bGetContentResult:= fbRequest.GetContent(pContent:= ADR(sContent),
                                                     nContentSize:= SIZEOF(sContent),
                                                     bSetNullTermination:= TRUE);
            IF fbRequest.nStatusCode >= 200 AND fbRequest.nStatusCode < 300 THEN
                bGetJsonResult:= FALSE;
                jsonDoc:= fbRequest.GetJsonDomContent(fbJson);
                IF jsonDoc <> 0 THEN
                    ; //do something with the weather data
                    nValidResCount:= nValidResCount+1;
                    bError:= FALSE;
                END_IF
                nResCount:= nResCount+1;
            END_IF
        END_IF
        nState:= 0;
        bBusy:= FALSE;
        IF bError THEN
            nErrCount:= nErrCount+1;
        END_IF
    END_IF
END_CASE
```

6.1.2 PostmanEcho

Dieser Teil des Beispiels umfasst drei HTTP-Befehle (GET, POST, PUT) für die Kommunikation mit Postman Echo, wobei es sich um einen Test-Webservice für REST-Clients handelt. Zusätzlich ist auch ein Beispiel für eine GET-Anfrage mit Header-Authentifizierung enthalten.

Die Postman Echo API ist speziell für Entwickler vorgesehen, um HTTP-Funktionen zu testen, und bietet noch viel mehr Möglichkeiten, als in unserem Beispiel gezeigt werden. Beispielsweise können verschiedene Authentifizierungsmethoden mit Postman Echo getestet werden.

```
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. Der Funktionsbaustein wird gestartet, indem die Variable bGet im Hauptprogramm auf TRUE gesetzt wird.
2. Die steigende Flanke wird dann verwendet, um eine GET-Anfrage mit dem Funktionsbaustein IotHttpRequest zu senden.

3. Nachdem die Anfrage abgeschlossen ist, wird die Fehlerbehandlung durchlaufen. Wenn weder der Funktionsbaustein selbst noch der HTTP-Statuscode einen Fehler aufweisen, wird die JSON-Antwort vom Webserver geparst.

```
FUNCTION_BLOCK FB_TestHTTP_Get
VAR_INPUT
    bSend               : BOOL;
END_VAR
VAR_IN_OUT
    fbClient            : FB_IotHttpClient;
END_VAR
VAR_OUTPUT
    bBusy               : BOOL;
    bError              : BOOL;
END_VAR
VAR
    fbRequest           : FB_IotHttpRequest;
    fbJson              : FB_JsonDomParser;
    nState              : UDINT;
    RisingEdge          : R_TRIG;

    bGetContentResult   : BOOL;
    sContent            : STRING(511);

    bGetJsonResult      : BOOL;
    jsonDoc             : SJsonValue;
    jsonVal             : SJsonValue;
    sResultValue        : STRING;

    nReqCount           : UDINT;
    nResCount           : UDINT;
    nValidResCount      : UDINT;
    nErrCount           : UDINT;
END_VAR
```

```
RisingEdge(CLK:= bSend);
CASE nState OF
0:
    IF RisingEdge.Q THEN
        IF fbRequest.SendRequest(sUri:= '/get?foo1=bar1&foo2=bar2',
                                 fbClient:= fbClient,
                                 eRequestType:= ETcIotHttpRequestType.HTTP_GET, 0, 0, 0) THEN
            nState:= 1;
            nReqCount:= nReqCount+1;
            bBusy:= TRUE;
            bError:= FALSE;
        END_IF
    END_IF
1:
    IF NOT fbRequest.bBusy THEN
        bError:= TRUE;
        IF NOT fbRequest.bError THEN
            bGetContentResult:= fbRequest.GetContent(pContent:= ADR(sContent),
                                                     nContentSize:= SIZEOF(sContent),
                                                     bSetNullTermination:= TRUE);
            IF fbRequest.nStatusCode >= 200 AND fbRequest.nStatusCode < 300 THEN
                bGetJsonResult:= FALSE;
                jsonDoc:= fbRequest.GetJsonDomContent(fbJson);
                IF jsonDoc <>0 THEN
                    bGetJsonResult:= TRUE;
                    IF fbJson.HasMember(jsonDoc, 'args') THEN
                        jsonVal:= fbJson.FindMember(jsonDoc, 'args');
                        IF fbJson.HasMember(jsonVal, 'foo2') THEN
                            jsonVal:= fbJson.FindMember(jsonVal, 'foo2');
                            nValidResCount:= nValidResCount+1;
                            bError:= FALSE;
                            IF fbJson.IsString(jsonVal) THEN
                                sResultValue:= fbJson.GetString(jsonVal);
                            END_IF
                        END_IF
                    END_IF
                END_IF
                nResCount:= nResCount+1;
            END_IF
        END_IF
        nState:= 0;
        bBusy:= FALSE;
        IF bError THEN
            nErrCount:= nErrCount+1;
```

```
        END_IF
    END_IF
END_CASE
```

6.1.2.2 Post

1. Der Funktionsbaustein wird gestartet, indem die Variable bPost im Hauptprogramm auf TRUE gesetzt wird.
2. Die steigende Flanke wird dann verwendet, um eine POST-Anfrage mit dem Funktionsbaustein IotHttpRequest zu senden.
3. Nachdem die Anfrage abgeschlossen ist, wird die Fehlerbehandlung durchlaufen. Wenn weder der Funktionsbaustein selbst noch der HTTP-Statuscode einen Fehler aufweisen, wird die JSON-Antwort vom Webserver geparst. Die JSON-Antwort enthält die Variable nReqCount, die an den Webserver gesendet wurde.

```
FUNCTION_BLOCK FB_TestHTTP_Post
VAR_INPUT
    bSend               : BOOL;
END_VAR
VAR_IN_OUT
    fbClient            : FB_IotHttpClient;
END_VAR
VAR_OUTPUT
    bBusy               : BOOL;
    bError              : BOOL;
END_VAR
VAR
    fbRequest           : FB_IotHttpRequest;
    fbJson              : FB_JsonDomParser;
    nState              : UDINT;
    RisingEdge          : R_TRIG;

    bGetContentResult   : BOOL;
    sContent            : STRING(511);

    bGetJsonResult      : BOOL;
    jsonDoc             : SJsonValue;
    jsonVal             : SJsonValue;
    sResultValue        : STRING;

    nReqCount           : UDINT;
    nResCount           : UDINT;
    nValidResCount      : UDINT;
    nErrCount           : UDINT;
END_VAR
```

```
RisingEdge(CLK:= bSend);
CASE nState OF
0:
    IF RisingEdge.Q THEN
        sContent:= UDINT_TO_STRING(nReqCount+1);
        IF fbRequest.SendRequest(sUri:= '/post?hello=world', fbClient:= fbClient,
                                 eRequestType:= ETcIotHttpRequestType.HTTP_POST,
                                 pContent:= ADR(sContent),
                                 nContentSize:= LEN2(ADR(sContent)), 0) THEN
            nState:= 1;
            nReqCount:= nReqCount+1;
            bBusy:= TRUE;
            bError:= FALSE;
        END_IF
    END_IF
1:
    IF NOT fbRequest.bBusy THEN
        bError:= TRUE;
        IF NOT fbRequest.bError THEN
            bGetContentResult:= fbRequest.GetContent(pContent:= ADR(sContent),
                                                     nContentSize:= SIZEOF(sContent),
                                                     bSetNullTermination:= TRUE);
            IF fbRequest.nStatusCode >= 200 AND fbRequest.nStatusCode < 300 THEN
                bGetJsonResult:= FALSE;
                jsonDoc:= fbRequest.GetJsonDomContent(fbJson);
                IF jsonDoc <>0 THEN
                    bGetJsonResult:= TRUE;
                    IF fbJson.HasMember(jsonDoc, 'data') THEN
                        jsonVal:= fbJson.FindMember(jsonDoc, 'data');
                        sResultValue:= fbJson.GetString(jsonVal);
```

```
                        IF STRING_TO_UDINT(sResultValue)= nReqCount THEN
                            nValidResCount:= nValidResCount+1;
                            bError:= FALSE;
                        END_IF
                    END_IF
                END_IF
                nResCount:= nResCount+1;
            END_IF
        END_IF
        nState:= 0;
        bBusy:= FALSE;
        IF bError THEN
            nErrCount:= nErrCount+1;
        END_IF
    END_IF
END_CASE
```

6.1.2.3 Put

1. Der Funktionsbaustein wird gestartet, indem die Variable bPut im Hauptprogramm auf TRUE gesetzt wird.
2. Die steigende Flanke wird dann verwendet, um eine PUT-Anfrage mit dem Funktionsbaustein IotHttpRequest zu senden.
3. Nachdem die Anfrage abgeschlossen ist, wird die Fehlerbehandlung durchlaufen. Wenn weder der Funktionsbaustein selbst noch der HTTP-Statuscode einen Fehler aufweisen, wird die JSON-Antwort vom Webserver geparst. Die JSON-Antwort enthält die Variable nReqCount, die an den Webserver gesendet wurde.

```
FUNCTION_BLOCK FB_TestHTTP_Put
VAR_INPUT
    bSend               : BOOL;
END_VAR
VAR_IN_OUT
    fbClient            : FB_IotHttpClient;
END_VAR
VAR_OUTPUT
    bBusy               : BOOL;
    bError              : BOOL;
END_VAR
VAR
    fbRequest           : FB_IotHttpRequest;
    fbJson              : FB_JsonDomParser;
    nState              : UDINT;
    RisingEdge          : R_TRIG;

    bGetContentResult   : BOOL;
    sContent            : STRING(511);

    bGetJsonResult      : BOOL;
    jsonDoc             : SJsonValue;
    jsonVal             : SJsonValue;
    sResultValue        : STRING;

    nReqCount           : UDINT;
    nResCount           : UDINT;
    nValidResCount      : UDINT;
    nErrCount           : UDINT;
END_VAR
```

```
RisingEdge(CLK:= bSend);
CASE nState OF
0:
    IF RisingEdge.Q THEN
        sContent:= UDINT_TO_STRING(nReqCount+1);
        IF fbRequest.SendRequest(sUri:= '/put', fbClient:= fbClient,
                                 eRequestType:= ETcIotHttpRequestType.HTTP_PUT,
                                 pContent:= ADR(sContent),
                                 nContentSize:= LEN2(ADR(sContent)), 0) THEN
            nState:= 1;
            nReqCount:= nReqCount+1;
            bBusy:= TRUE;
            bError:= FALSE;
        END_IF
    END_IF
```

```
1:
    IF NOT fbRequest.bBusy THEN
        bError:= TRUE;
        IF NOT fbRequest.bError THEN
            bGetContentResult:= fbRequest.GetContent(pContent:= ADR(sContent),
                                                     nContentSize:= SIZEOF(sContent),
                                                     bSetNullTermination:= TRUE);
            IF fbRequest.nStatusCode >= 200 AND fbRequest.nStatusCode < 300 THEN
                bGetJsonResult:= FALSE;
                jsonDoc:= fbRequest.GetJsonDomContent(fbJson);
                IF jsonDoc <>0 THEN
                    bGetJsonResult:= TRUE;
                    IF fbJson.HasMember(jsonDoc, 'data') THEN
                        jsonVal:= fbJson.FindMember(jsonDoc, 'data');
                        sResultValue:= fbJson.GetString(jsonVal);
                        IF STRING_TO_UDINT(sResultValue)= nReqCount THEN
                            nValidResCount:= nValidResCount+1;
                            bError:= FALSE;
                        END_IF
                    END_IF
                END_IF
                nResCount:= nResCount+1;
            END_IF
        END_IF
        nState:= 0;
        bBusy:= FALSE;
        IF bError THEN
            nErrCount:= nErrCount+1;
        END_IF
    END_IF
END_CASE
```

6.1.2.4 Header-Authentifizierung

1. Der Funktionsbaustein wird gestartet, indem die Variable bHeaderAuth im Hauptprogramm auf TRUE gesetzt wird.
2. Die steigende Flanke wird dann verwendet, um eine GET-Anfrage mit dem Funktionsbaustein IotHttpRequest zu senden. Diese Anfrage enthält ein zusätzliches Header-Feld für die Authentifizierung.
3. Nachdem die Anfrage abgeschlossen ist, wird die Fehlerbehandlung durchlaufen. Wenn weder der Funktionsbaustein selbst noch der HTTP-Statuscode einen Fehler aufweisen, wird die JSON-Antwort vom Webserver geparst. Die JSON-Antwort enthält die Information, ob die Authentifizierung erfolgreich war.

```
FUNCTION_BLOCK FB_TestHTTP_HeaderAuth
VAR_INPUT
    bSend               : BOOL;
END_VAR
VAR_IN_OUT
    fbClient            : FB_IotHttpClient;
END_VAR
VAR_OUTPUT
    bBusy               : BOOL;
    bError              : BOOL;
END_VAR
VAR
    fbRequest           : FB_IotHttpRequest;
    fbHeader            : FB_IotHttpHeaderFieldMap;
    fbJson              : FB_JsonDomParser;
    nState              : UDINT;
    RisingEdge          : R_TRIG;

    bGetContentResult   : BOOL;
    sContent            : STRING(511);

    bGetJsonResult      : BOOL;
    jsonDoc             : SJsonValue;
    jsonVal             : SJsonValue;
    bResultValue        : BOOL;

    nReqCount           : UDINT;
    nResCount           : UDINT;
    nValidResCount      : UDINT;
    nErrCount           : UDINT;
    bOnce               : BOOL:= TRUE;
END_VAR
```

```
IF bOnce THEN
    fbHeader.AddField('Authorization', 'Basic cG9zdG1hbjpwYXNzd29yZA==', FALSE);
    bOnce:= FALSE;
END_IF

RisingEdge(CLK:= bSend);
CASE nState OF
0:
    IF RisingEdge.Q THEN
        IF fbRequest.SendRequest(sUri:= '/basic-auth', fbClient:= fbClient,
                                 eRequestType:= ETcIotHttpRequestType.HTTP_GET, 0, 0, fbHeader) THEN
            nState:= 1;
            nReqCount:= nReqCount+1;
            bBusy:= TRUE;
            bError:= FALSE;
        END_IF
    END_IF
1:
    IF NOT fbRequest.bBusy THEN
        bError:= TRUE;
        IF NOT fbRequest.bError THEN
            bGetContentResult:= fbRequest.GetContent(pContent:= ADR(sContent),
                                                     nContentSize:= SIZEOF(sContent),
                                                     bSetNullTermination:= TRUE);
            IF fbRequest.nStatusCode >= 200 AND fbRequest.nStatusCode < 300 THEN
                bGetJsonResult:= FALSE;
                jsonDoc:= fbRequest.GetJsonDomContent(fbJson);
                IF jsonDoc <>0 THEN
                    bGetJsonResult:= TRUE;
                    IF fbJson.HasMember(jsonDoc, 'authenticated') THEN
                        jsonVal:= fbJson.FindMember(jsonDoc, 'authenticated');
                        IF fbJson.IsBool(jsonVal) THEN
                            bResultValue:= fbJson.GetBool(jsonVal);
                            nValidResCount:= nValidResCount+1;
                            bError:= FALSE;
                        END_IF
                    END_IF
                END_IF
                nResCount:= nResCount+1;
            END_IF
        END_IF
        nState:= 0;
        bBusy:= FALSE;
        IF bError THEN
            nErrCount:= nErrCount+1;
        END_IF
    END_IF
END_CASE
```

6.1.3 AWS IoT

Dieses Beispiel zeigt zwei verschiedene Szenarios mit AWS IoT. Eine GET-Anfrage an einen IoT Shadow und eine POST-Anfrage an den AWS IoT Core-Message-Broker werden dargestellt.

Die GET-Anfrage ähnelt dem Beispiel „OpenWeatherMap“, es wurde lediglich der Uri in den Uri des Shadows des IoT Thing im Hinblick auf die AWS REST-API geändert.

Der Benutzer muss ein eigenes AWS-Konto haben, damit ein IoT Core-Endpunkt vorhanden ist und die Möglichkeit besteht, Zertifikate für die Kommunikation mit TwinCAT zu erstellen.

```
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. Der Funktionsbaustein wird gestartet, indem die Variable bGetAwsIotShadow im Hauptprogramm auf TRUE gesetzt wird.
2. Die steigende Flanke wird dann verwendet, um eine GET-Anfrage mit dem Funktionsbaustein IotHttpRequest zu senden.
3. Nachdem die Anfrage abgeschlossen ist, wird die Fehlerbehandlung durchlaufen. Wenn weder der Funktionsbaustein selbst noch der HTTP-Statuscode einen Fehler aufweisen, kann die JSON-Antwort vom Webserver im SPS-Programm geparst werden.

```
FUNCTION_BLOCK FB_TestHTTP_Get_awsIotShadow
VAR_INPUT
    bSend               : BOOL;
END_VAR
VAR_IN_OUT
    fbClient            : FB_IotHttpClient;
END_VAR
VAR_OUTPUT
    bBusy               : BOOL;
    bError              : BOOL;
END_VAR
VAR
    fbRequest           : FB_IotHttpRequest;
    fbJson              : FB_JsonDomParser;
    nState              : UDINT;
    RisingEdge          : R_TRIG;

    bGetContentResult   : BOOL;
    sContent            : STRING(511);

    bGetJsonResult      : BOOL;
    jsonDoc             : SJsonValue;
    jsonVal             : SJsonValue;
    sResultValue        : STRING;

    nReqCount           : UDINT;
    nResCount           : UDINT;
    nValidResCount      : UDINT;
    nErrCount           : UDINT;
END_VAR
```

```
RisingEdge(CLK:= bSend);
CASE nState OF
0:
    IF RisingEdge.Q THEN
        IF fbRequest.SendRequest(sUri:= '/things/thingName/shadow', fbClient:= fbClient,
                                 eRequestType:= ETcIotHttpRequestType.HTTP_GET, 0, 0, 0) THEN
            nState:= 1;
            nReqCount:= nReqCount+1;
            bBusy:= TRUE;
            bError:= FALSE;
        END_IF
    END_IF
1:
    IF NOT fbRequest.bBusy THEN
        bError:= TRUE;
        IF NOT fbRequest.bError THEN
            bGetContentResult:= fbRequest.GetContent(pContent:= ADR(sContent),
                                                     nContentSize:= SIZEOF(sContent),
                                                     bSetNullTermination:= TRUE);
            IF fbRequest.nStatusCode >= 200 AND fbRequest.nStatusCode < 300 THEN
                bGetJsonResult:= FALSE;
                jsonDoc:= fbRequest.GetJsonDomContent(fbJson);
                IF jsonDoc <>0 THEN
                    ; // do something with the shadow document
                    nValidResCount:= nValidResCount+1;
```

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

6.1.3.2 Post

1. Der Funktionsbaustein wird gestartet, indem die Variable bPostAwsIot im Hauptprogramm auf TRUE gesetzt wird.
2. Die steigende Flanke wird dann verwendet, um eine POST-Anfrage mit dem Funktionsbaustein IotHttpRequest zu senden.
3. Nachdem die Anfrage abgeschlossen ist, wird die Fehlerbehandlung durchlaufen. Wenn weder der Funktionsbaustein selbst noch der HTTP-Statuscode einen Fehler aufweisen, wird die JSON-Antwort vom Webserver geparst. Die JSON-Antwort enthält die Nachricht „OK“, wenn die POST-Anfrage erfolgreich war.

```
FUNCTION_BLOCK FB_TestHTTP_Post_awsIot
VAR_INPUT
    bSend               : BOOL;
END_VAR
VAR_IN_OUT
    fbClient            : FB_IotHttpClient;
END_VAR
VAR_OUTPUT
    bBusy               : BOOL;
    bError              : BOOL;
END_VAR
VAR
    fbRequest           : FB_IotHttpRequest;
    fbJson              : FB_JsonDomParser;
    fbPayload           : FB_JsonSaxWriter;
    nState              : UDINT;
    RisingEdge          : R_TRIG;

    bGetContentResult   : BOOL;
    sContent            : STRING(511);

    bGetJsonResult      : BOOL;
    jsonDoc             : SJsonValue;
    jsonVal             : SJsonValue;
    sResultValue        : STRING;

    nReqCount           : UDINT;
    nResCount           : UDINT;
    nValidResCount      : UDINT;
    nErrCount           : UDINT;
END_VAR
```

```
RisingEdge(CLK:= bSend);
CASE nState OF
0:
    IF RisingEdge.Q THEN
        fbPayload.StartObject();
        fbPayload.AddKey('message');
        fbPayload.AddReal(42.42);
        fbPayload.EndObject();
        sContent:= fbPayload.GetDocument();
        fbPayload.ResetDocument();
        IF fbRequest.SendRequest(sUri:= '/topics/mytopic?qos=1', fbClient:= fbClient,
                                 eRequestType:= ETcIotHttpRequestType.HTTP_POST,
                                 pContent:= ADR(sContent),
                                 nContentSize:= LEN2(ADR(sContent)), 0) THEN
            nState:= 1;
            nReqCount:= nReqCount+1;
            bBusy:= TRUE;
            bError:= FALSE;
        END_IF
    END_IF
```

```
1:
    IF NOT fbRequest.bBusy THEN
        bError:= TRUE;
        IF NOT fbRequest.bError THEN
            bGetContentResult:= fbRequest.GetContent(pContent:= ADR(sContent),
                                                     nContentSize:= SIZEOF(sContent),
                                                     bSetNullTermination:= TRUE);
            IF fbRequest.nStatusCode >= 200 AND fbRequest.nStatusCode < 300 THEN
                bGetJsonResult:= FALSE;
                jsonDoc:= fbRequest.GetJsonDomContent(fbJson);
                IF jsonDoc <>0 THEN
                    bGetJsonResult:= TRUE;
                    IF fbJson.HasMember(jsonDoc, 'message') THEN
                        jsonVal:= fbJson.FindMember(jsonDoc, 'message');
                        sResultValue:= fbJson.GetString(jsonVal);
                        IF sResultValue= 'OK' THEN
                            nValidResCount:= nValidResCount+1;
                            bError:= FALSE;
                        END_IF
                    END_IF
                END_IF
                nResCount:= nResCount+1;
            END_IF
        END_IF
        nState:= 0;
        bBusy:= FALSE;
        IF bError THEN
            nErrCount:= nErrCount+1;
        END_IF
    END_IF
END_CASE
```

6.1.4 Philips Hue

Dieser Teil des Beispiels umfasst eine PUT-Anfrage an eine Philips Hue Bridge (Erforderliche TwinCAT-Version: 3.1.4024.10). Die REST-API der Bridge ermöglicht es dem TwinCAT-Benutzer, Philips Hue Geräte aus der SPS heraus zu steuern. Der Benutzer benötigt dazu eine Philips Hue Bridge, die in seinem Netzwerk verfügbar ist, und mindestens ein Gerät, das damit verbunden ist.

Im Grunde sendet der HTTP-Client, in diesem Fall TwinCAT, verschiedene Werte als JSON-Dokument an die Bridge (z. B. Sättigung, Helligkeit, Zustands- und Farbwert für eine Glühbirne). Die Bridge antwortet mit einem anderen JSON-Dokument, das zeigt, ob die Befehle erfolgreich waren.

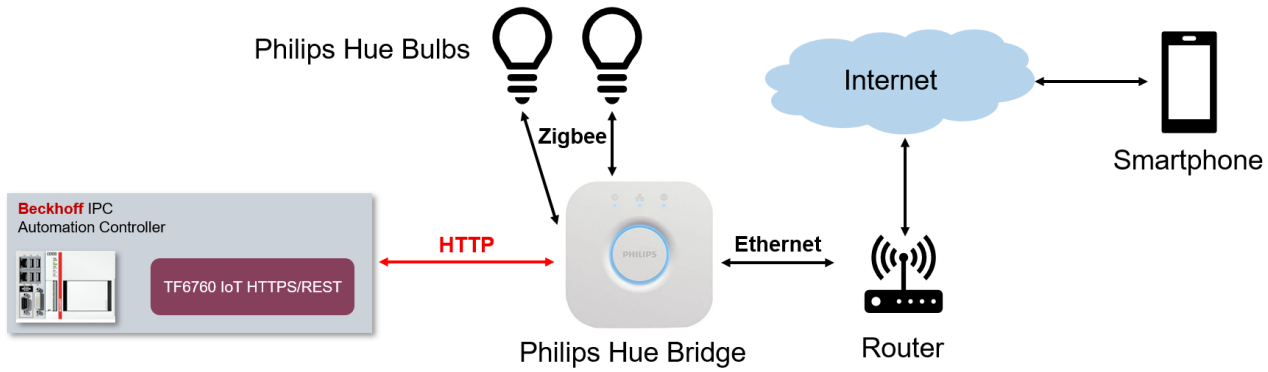


Abb. 3: Schema Philips Hue

Beim Setup-Prozess generiert die Philips Hue Bridge zufällig einen Benutzernamen für ein Gerät, das als HTTP-Client fungiert. Dieser Benutzername kann dann für die Kommunikation mit der Bridge verwendet werden.

Philips bietet hier ein Tutorial „Erste Schritte“, in dem erläutert wird, wie die Bridge in einem Netzwerk zu finden ist und wie eine Client-Anwendung implementiert wird: https://developers.meethue.com/develop/get-started-2/
Die API-Beschreibung ist auf derselben Website zu finden, hierfür ist eine Registrierung erforderlich.

Das folgende Beispiel implementiert zum einen eine einfache PUT-Anfrage und zum anderen eine toggelnde PUT-Anfrage, die für einen sogenannten „Blinkmodus“ verwendet wird. Dieser Modus veranlasst eine Philips Hue Go Lampe dazu, jedes Mal, wenn der Timer-Ausgang gesetzt wird, die Farbe zu wechseln (zwischen 0 und 65000).

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

    fbHttpClientPhilips Hue        : FB_IotHttpClient :=(sHostName:='172.17.x.x',
                                     bKeepAlive:=TRUE, tConnectionTimeout:=T#10S);

    fbHttpPutPhilipsHue            : FB_TestHTTP_Put_PhilipsHue;

    bBlinkingMode                  : BOOL;
    fbTimer                        : TON:=(PT:=T#500MS);
    nColor                         : UINT;
END_VAR
```

```
//init client parameters at startup
IF NOT fbHttpClientPhilipsHue.bConfigured THEN
    fbHttpClientPhilipsHue.nHostPort:= 80;
    fbHttpClientPhilipsHue.stTLS.bNoServerCertCheck:= FALSE;
END_IF

IF fbHttpClientPhilipsHue.bConfigured THEN
    fbHttpPutPhilipsHue(bSend:=bPutPhilipsHue, fbClient:=fbHttpClientPhilipsHue, nColor:=nColor);
END_IF

IF bBlinkingMode THEN
    fbTimer(IN:=NOT fbTimer.Q);
    IF fbTimer.Q THEN
        nColor:=nColor+5000;
        IF nColor=65000 THEN
            nColor:=0;
        END_IF
        IF bPutPhilipsHue THEN
            bPutPhilipsHue:=FALSE;
        ELSE
            bPutPhilipsHue:=TRUE;
        END_IF
    END_IF
END_IF

fbHttpClientPhilipsHue.Execute();
```

6.1.4.1 Put

1. Der Funktionsbaustein wird gestartet, indem die Variable bPutPhilipsHue im Hauptprogramm auf TRUE gesetzt wird.
2. Die steigende Flanke wird dann verwendet, um eine PUT-Anfrage mit dem Funktionsbaustein IotHttpRequest zu senden.
3. Nachdem die Anfrage abgeschlossen ist, wird die Fehlerbehandlung durchlaufen. Wenn weder der Funktionsbaustein selbst noch der HTTP-Statuscode einen Fehler aufweisen, könnte die JSON-Antwort von der Philips Hue Bridge geparst werden. Die JSON-Antwort zeigt dem Benutzer, ob die Anfrage erfolgreich war.

```
FUNCTION_BLOCK FB_TestHTTP_Put_PhilipsHue
VAR_INPUT
    bSend               : BOOL;
    nColor              : UINT;
END_VAR
VAR_IN_OUT
    fbClient            : FB_IotHttpClient;
END_VAR
VAR_OUTPUT
    bBusy               : BOOL;
    bError              : BOOL;
END_VAR
VAR
    fbRequest           : FB_IotHttpRequest;
    fbJson              : FB_JsonDomParser;
    fbJsonWriter        : FB_JsonSaxWriter;
```

```
    nState                : UDINT;
    RisingEdge            : R_TRIG;

    bGetContentResult     : BOOL;
    sContent              : STRING(511);
    sSend                 : STRING(511);

    bGetJsonResult        : BOOL;
    jsonDoc               : SJsonValue;
    jsonVal               : SJsonValue;
    sResultValue          : STRING;

    nReqCount             : UDINT;
    nResCount             : UDINT;
    nValidResCount        : UDINT;
    nErrCount             : UDINT;

    fbTimer               : TON;
    bLightOn              : BOOL;
END_VAR
```

```
RisingEdge(CLK:= bSend );

CASE nState OF
0:
        fbJsonWriter.StartObject();
        fbJsonWriter.AddKey('on');
        fbJsonWriter.AddBool(TRUE);
        fbJsonWriter.AddKey('sat');
        fbJsonWriter.AddUdint(50);
        fbJsonWriter.AddKey('bri');
        fbJsonWriter.AddUdint(100);
        fbJsonWriter.AddKey('hue');
        fbJsonWriter.AddUdint(nColor);
        fbJsonWriter.EndObject();
        sSend:=fbJsonWriter.GetDocument();
        fbJsonWriter.ResetDocument();
    IF RisingEdge.Q THEN
        IF fbRequest.SendRequest(sUri:='/api/qiyut7ubQQQv2xKJfFe9cVvIroUGHtIk2eYsIfGX/lights/1/
state', fbClient:=fbClient, eRequestType:=ETcIotHttpRequestType.HTTP_PUT,
            pContent:=ADR(sSend), nContentSize:=LEN2(ADR(sSend)), 0) THEN
            nState:= 1;
            nReqCount:= nReqCount+1;
            bBusy:= TRUE;
            bError:= FALSE;
        END_IF
    END_IF
1:
    IF NOT fbRequest.bBusy THEN
        bError:= TRUE;
        IF NOT fbRequest.bError THEN
            bGetContentResult:= fbRequest.GetContent(pContent:= ADR(sContent), nContentSize:= SIZEOF
(sContent), bSetNullTermination:= TRUE);
            IF fbRequest.nStatusCode >= 200 AND fbRequest.nStatusCode < 300 THEN
                bGetJsonResult:= FALSE;
                jsonDoc:= fbRequest.GetJsonDomContent(fbJson);
                IF jsonDoc <> 0 THEN
                    ; // do something with status response
                    nValidResCount:= nValidResCount+1;
                    bError:= FALSE;
                END_IF
                nResCount:= nResCount+1;
            END_IF
        END_IF
        nState:= 0;
        bBusy:= FALSE;
        IF bError THEN
            nErrCount:= nErrCount+1;
        END_IF
    END_IF
END_CASE
```

6.1.5 Telegram

Dieser Teil des Beispiels umfasst eine GET-Anfrage an die REST-API des Telegram-Messengers (Erforderliche TwinCAT-Version: 3.1.4024.10). Mit Hilfe des sogenannten „Telegram-Bots“ kann ein TwinCAT-Benutzer Mitteilungen an einen bestimmten Telegram-Chat senden.

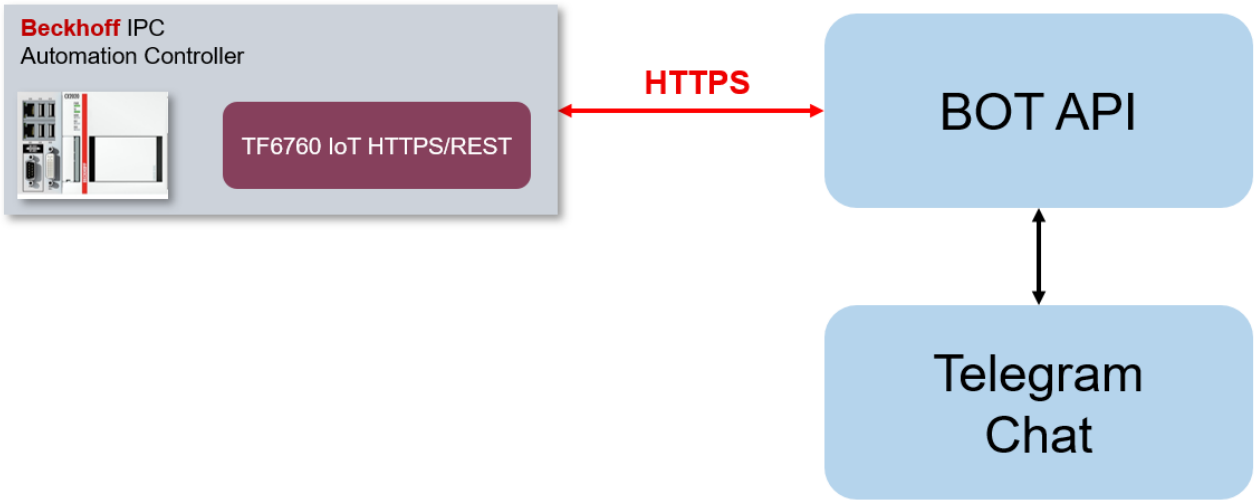


Abb. 4: Schema Telegram

Der erste Schritt der Herstellung einer Verbindung ist die Erstellung des „Telegram-Bots“. Dafür benötigt der Benutzer ein Telegram-Konto, von dem aus er den Bot kontaktieren kann.

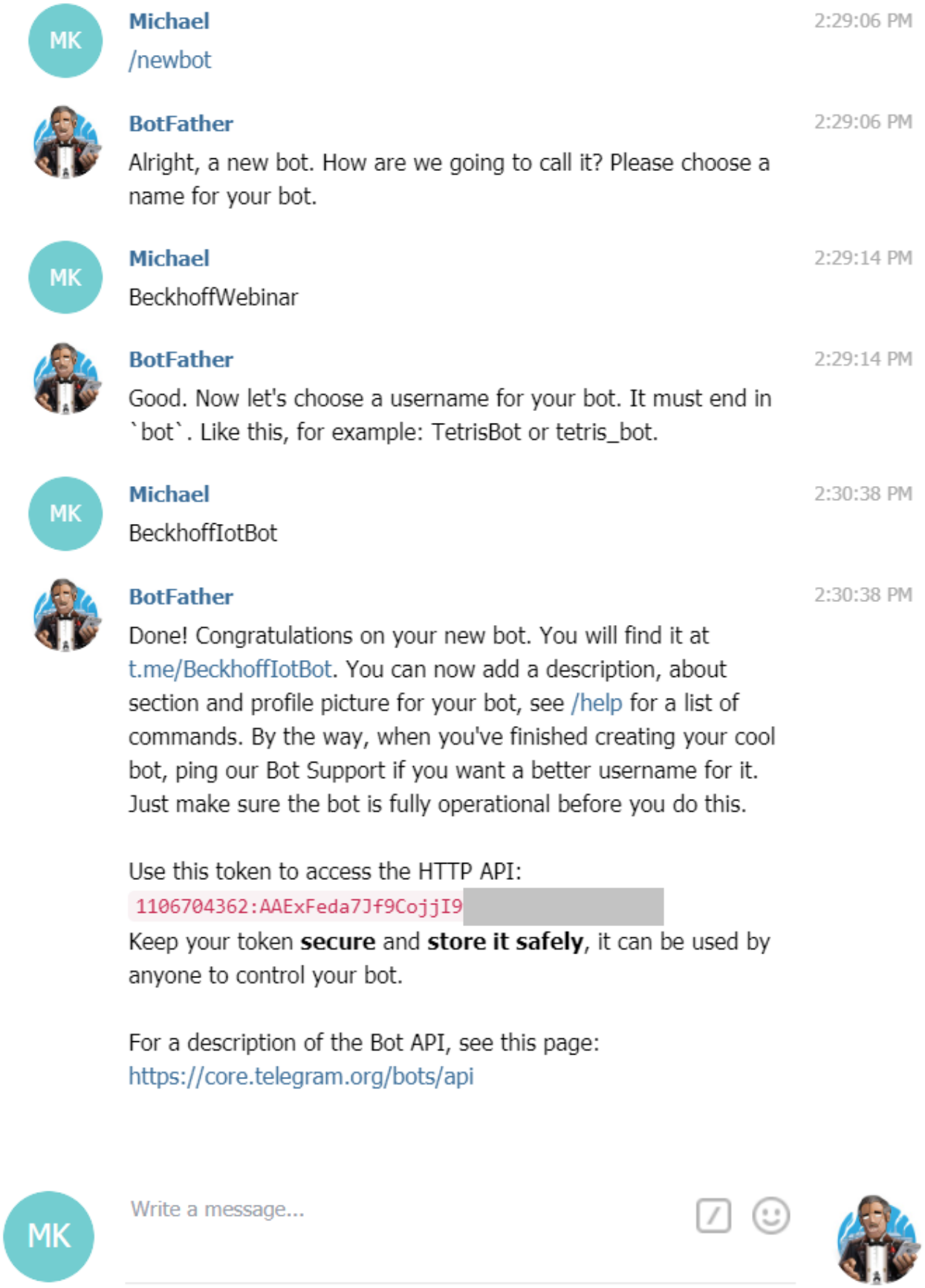


Abb. 5: Telegram BotFather

Der Bot liefert ein API-Token zurück, das beim Senden von Anfragen aus TwinCAT an die Telegram-API verwendet werden muss. Mit Zugriff auf dieses Token kann ein HTTP-Client jede Funktion steuern, die der Bot bietet. Eine Beschreibung der BOT-API findet sich hier: https://core.telegram.org/bots/api

Das Senden von Mitteilungen aus einer TwinCAT SPS an einen Telegram-Bot kann wie im folgenden Bild dargestellt ablaufen. In diesem Beispiel wird für die Anforderung der Chat-ID eine Mitteilung von der Telegram-Browserversion an den Bot verwendet.

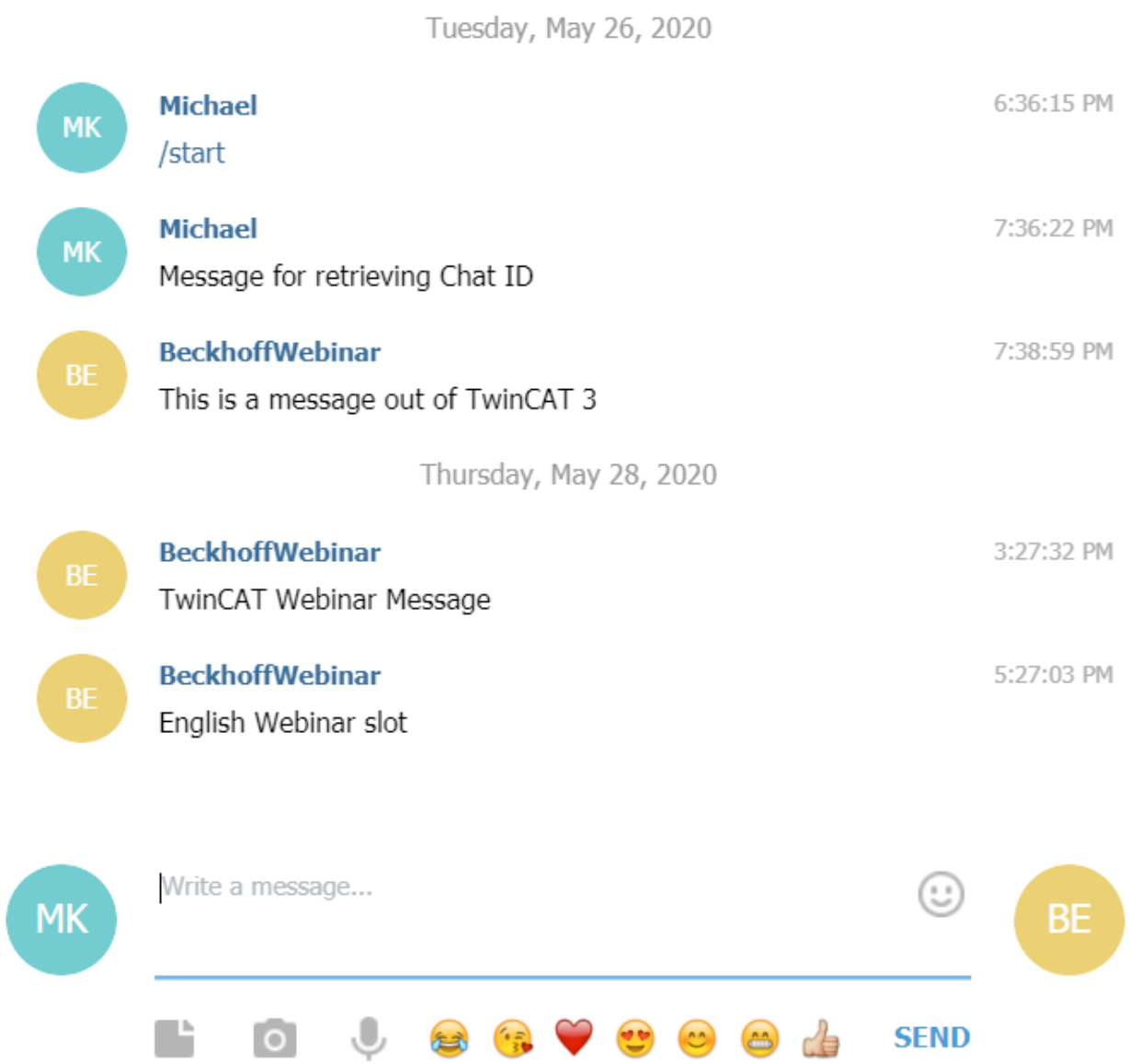


Abb. 6: Telegram-Chat

```
PROGRAM MAIN
VAR
    // trigger command execution for Telegram samples
    bGetTelegram                    : BOOL;

    fbHttpClientTelegram            : FB_IotHttpClient :=(sHostName:='api.telegram.org',
                                    bKeepAlive:=FALSE, tConnectionTimeout:=T#10S);

    fbHttpGetTelegram               : FB_TestHTTP_Get_Telegram;
    sMessage                        : STRING(500);
END_VAR
```

```
//init client parameters at startup
IF NOT fbHttpClientTelegram.bConfigured THEN
    fbHttpClientTelegram.stTLS.sCA:='C:\TwinCAT\3.1\Config\Certificates\TelegramRoot.cer';
    fbHttpClientTelegram.nHostPort:=443;
    fbHttpClientTelegram.stTLS.bNoServerCertCheck:=FALSE;
```

```
END_IF

IF fbHttpClientTelegram.bConfigured THEN
    fbHttpGetTelegram(bSend:=bGetTelegram, fbClient:=fbHttpClientTelegram, sMessage:=sMessage);
END_IF

fbHttpClientTelegram.Execute();
```

6.1.5.1 Get

1. Der Funktionsbaustein wird gestartet, indem die Variable bGetTelegram im Hauptprogramm auf TRUE gesetzt wird.
2. Die steigende Flanke wird dann verwendet, um eine GET-Anfrage mit dem Funktionsbaustein IotHttpRequest zu senden.
3. Nachdem die Anfrage abgeschlossen ist, wird die Fehlerbehandlung durchlaufen. Wenn weder der Funktionsbaustein selbst noch der HTTP-Statuscode einen Fehler aufweisen, kann die JSON-Antwort vom Webserver im SPS-Programm geparst werden.

```
FUNCTION_BLOCK FB_TestHTTP_Get_Telegram
VAR_INPUT
    bSend               : BOOL;
    sMessage            : STRING(500);
END_VAR
VAR_IN_OUT
    fbClient            : FB_IotHttpClient;
END_VAR
VAR_OUTPUT
    bBusy               : BOOL;
    bError              : BOOL;
END_VAR
VAR
    fbRequest           : FB_IotHttpRequest;
    fbJson              : FB_JsonDomParser;
    nState              : UDINT;
    RisingEdge          : R_TRIG;

    bGetContentResult   : BOOL;
    sContent            : STRING(511);

    bGetJsonResult      : BOOL;
    jsonDoc             : SJsonValue;
    jsonVal             : SJsonValue;
    sResultValue        : STRING;

    nReqCount           : UDINT;
    nResCount           : UDINT;
    nValidResCount      : UDINT;
    nErrCount           : UDINT;
    fbFormatString      : FB_FormatString;
    sBotApiKey          : STRING(500):='1106704362:AAExFeda7Jf9CojjI9whLEzPeE4KDlDzEnf';
    sChatId             : STRING(500):='1140427258';
    sConMessage         : STRING(500);
END_VAR
RisingEdge(CLK:= bSend );
```

```
fbFormatString(
    sFormat := '/bot%s/sendMessage?chat_id=%s&text=%s',
    arg1    := F_STRING(sBotApiKey),
    arg2    := F_STRING(sChatId),
    arg3    := F_STRING(sMessage),
    bError  => ,
    nErrId  => ,
    sOut    => sConMessage);

CASE nState OF
0:
    IF RisingEdge.Q THEN
        IF fbRequest.SendRequest(sUri:=sConMessage, fbClient:=fbClient, eRequestType:=ETcIotHttpRequ
estType.HTTP_Get, 0, 0, 0) THEN
            nState:= 1;
            nReqCount:= nReqCount+1;
            bBusy:= TRUE;
            bError:= FALSE;
        END_IF
```

```
    END_IF
1:
    IF NOT fbRequest.bBusy THEN
        bError:= TRUE;
        IF NOT fbRequest.bError THEN
            bGetContentResult:= fbRequest.GetContent(pContent:= ADR(sContent), nContentSize:= SIZEOF
(sContent), bSetNullTermination:= TRUE);
            IF fbRequest.nStatusCode >= 200 AND fbRequest.nStatusCode < 300 THEN
                bGetJsonResult:= FALSE;
                jsonDoc:= fbRequest.GetJsonDomContent(fbJson);
                IF jsonDoc <> 0 THEN
                    ; // do something with the response
                    nValidResCount:= nValidResCount+1;
                    bError:= FALSE;
                END_IF
                nResCount:= nResCount+1;
            END_IF
        END_IF
        nState:= 0;
        bBusy:= FALSE;
        IF bError THEN
            nErrCount:= nErrCount+1;
        END_IF
    END_IF
END_CASE
```

6.1.6 AWS Service Configuration

In diesem Beispiel wird aufgezeigt, wie aus der SPS mithilfe einer GET-Anfrage auf Services des Cloud-Anbieters Amazon Web Services (AWS) zugegriffen werden kann (Erforderliche TwinCAT-Version: 3.1.4024.12). Die REST API liefert alle Funktionen, die ein Benutzer auch in der Management-Konsole von AWS hat. Es können beispielsweise über den AWS EC2-Service virtuelle Maschinen instanziiert oder auch konfiguriert werden.

Das Kapitel AWS Signature Version 4 [▶ 20] reißt kurz den Hintergrund der AWS-eigenen Signierungsfunktion an. Alles andere entnehmen Sie bitte der AWS-Dokumentation.

Wie bereits im Kapitel URL-Redirects [▶ 20] beschrieben, wertet der IoT-Treiber keine URL-Um- und Weiterleitungen aus. Deshalb muss beim Zugriff auf den lokalen Endpoint eines AWS-Rechenzentrums immer die exakte Adresse angegeben werden. Wieder auf das Beispiel der EC2-Services bezogen bedeutet dies, dass sich ein Benutzer aus TwinCAT nicht mit *ec2.amazonaws.com* verbinden kann, sondern direkt die Region im Link mitgeben muss: *ec2.eu-central-1.amazonaws.com*.

Ein mögliches Vorgehen hier wäre: Alle verfügbaren Regionen über eine ortsunabhängige AWS-REST API-Funktion abholen und daraus den Region Endpoint für die gewünschte Region extrahieren. Dies hätte gegen eine statische Programmierung des Endpoints den Vorteil, dass Änderungen in der Endpoint-URL einer Region keine Änderungen im Programmcode nach sich ziehen würden.

```
PROGRAM MAIN
VAR
    // trigger command execution for AWS Sig V4 samples
    bGetAWSSigV4                    : BOOL;

    fbHttpClientAWSSigV4            : FB_IotHttpClient :=(sHostName:='ec2.us-east-1.amazonaws.com',
                                    bKeepAlive:=FALSE, tConnectionTimeout:=T#10S);

    fbHttpGetAWSSigV4               : FB_TestHTTP_Get_AwsSigV4;
END_VAR
```

```
//init client parameters at startup
IF NOT fbHttpClientAWSSigV4.bConfigured THEN
    fbHttpClientAWSSigV4.nHostPort:=443;
    fbHttpClientAWSSigV4.stTLS.bNoServerCertCheck:=TRUE;
END_IF

IF fbHttpClientAWSSigV4.bConfigured THEN
    fbHttpGetAWSSigV4(bSend:= bGetAWSSigV4, fbClient:= fbHttpClientAWSSigV4);
END_IF

fbHttpClientAWSSigV4.Execute();
```

6.1.6.1 Get

1. Der Funktionsbaustein wird gestartet, indem die Variable bGetAWSSigV4 im Hauptprogramm auf TRUE gesetzt wird.
2. Die steigende Flanke wird dann verwendet, um eine GET-Anfrage mit dem Funktionsbaustein IotHttpRequest zu senden.
3. Nachdem die Anfrage abgeschlossen ist, wird die Fehlerbehandlung durchlaufen. Wenn weder der Funktionsbaustein selbst noch der HTTP-Statuscode einen Fehler aufweisen, kann die XML-Antwort vom Webserver im SPS-Programm geparst werden (Die Antwort der AWS-REST API kann ziemlich lang werden, deshalb reservieren Sie genug Speicherplatz und probieren Sie es ggf. vorher mit einem HTTP-Testclient aus).

Alphabetische Sortierung notwendig

Die Request-URL muss ebenso wie der signierte Header alphabetisch sortiert werden. Bei dem signierten Header übernimmt dies der IoT-Treiber, bei der Request-URL muss der Be-nutzer selbst darauf achten, ansonsten kommt eine Fehlermeldung vom HTTP-Server.

```
FUNCTION_BLOCK FB_TestHTTP_Get_AWSSigV4
VAR_INPUT
    bSend               : BOOL;
END_VAR
VAR_IN_OUT
    fbClient            : FB_IotHttpClient;
END_VAR
VAR_OUTPUT
    bBusy               : BOOL;
    bError              : BOOL;
END_VAR
VAR
    fbRequest           : FB_IotHttpRequest;
    fbSig4Header        : FB_IotHttpAwsSigV4HeaderFieldMap;
    nState              : UDINT;
    RisingEdge          : R_TRIG;

    bGetContentResult   : BOOL;
    sContent            : STRING(5000);

    bGetJsonResult      : BOOL;
    sResultValue        : STRING;

    nReqCount           : UDINT;
    nResCount           : UDINT;
    nValidResCount      : UDINT;
    nErrCount           : UDINT;
    sRequestUrl         : STRING(500):='?
Action=RunInstances&ImageId=ami-0229f7666f517b31e&InstanceType=t2.small&KeyName=TestKDB&MaxCount=1&M
inCount=1&Version=2016-11-15';
    //sRequestUrl       : STRING(500):='?Action=DescribeInstances&Version=2016-11-15';
    sService            : STRING:='ec2';
    sRegion             : STRING:='us-east-1';
    sAccessKey          : STRING:='censored';
    sSecretKey          : STRING:='censored';
    sSignedHeaders      : STRING:='host;x-amz-date';
    bSetParameter       : BOOL;
END_VAR
```

```
RisingEdge(CLK:= bSend );

CASE nState OF
0:
    IF RisingEdge.Q THEN
    bSetParameter:=fbSig4Header.SetParameter(sService, sRegion, sAccessKey, sSecretKey, sSignedHeade
rs);
        IF fbRequest.SendRequest(sUri:=sConMessage, fbClient:=fbClient, eRequestType:=ETcIotHttpRequ
estType.HTTP_Get, 0, 0, fbHeader:=fbSig4Header) THEN
            nState:= 1;
            nReqCount:= nReqCount+1;
            bBusy:= TRUE;
            bError:= FALSE;
        END_IF
        bSetParameter:=FALSE;
    END_IF
1:
    IF NOT fbRequest.bBusy THEN
```

```
        bError:= TRUE;
        IF NOT fbRequest.bError THEN
            bGetContentResult:= fbRequest.GetContent(pContent:= ADR(sContent), nContentSize:= SIZEOF
(sContent), bSetNullTermination:= TRUE);
            IF fbRequest.nStatusCode >= 200 AND fbRequest.nStatusCode < 300 THEN
                nResCount:= nResCount+1;
                // do something with the XML response
                bError:=FALSE;
            END_IF
        END_IF
        nState:= 0;
        bBusy:= FALSE;
        IF bError THEN
            nErrCount:= nErrCount+1;
        END_IF
    END_IF
END_CASE
```

7 Anhang

7.1 ADS Return Codes

Gruppierung der Fehlercodes:
Globale Fehlercodes: 0x0000 [▶ 146]... (0x9811_0000 ...)
Router Fehlercodes: 0x0500 [▶ 146]... (0x9811_0500 ...)
Allgemeine ADS Fehler: 0x0700 [▶ 147]... (0x9811_0700 ...)
RTime Fehlercodes: 0x1000 [▶ 149]... (0x9811_1000 ...)

Globale Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x0	0	0x98110000	ERR_NOERROR	Kein Fehler.
0x1	1	0x98110001	ERR_INTERNAL	Interner Fehler.
0x2	2	0x98110002	ERR_NORTIME	Keine Echtzeit.
0x3	3	0x98110003	ERR_ALLOCLOCKEDMEM	Zuweisung gesperrt - Speicherfehler.
0x4	4	0x98110004	ERR_INSERTMAILBOX	Postfach voll – Es konnte die ADS Nachricht nicht versendet werden. Reduzieren der Anzahl der ADS Nachrichten pro Zyklus bringt Abhilfe.
0x5	5	0x98110005	ERR_WRONGRECEIVEHMSG	Falsches HMSG.
0x6	6	0x98110006	ERR_TARGETPORTNOTFOUND	Ziel-Port nicht gefunden – ADS Server ist nicht gestartet oder erreichbar.
0x7	7	0x98110007	ERR_TARGETMACHINENOTFOUND	Zielrechner nicht gefunden – AMS Route wurde nicht gefunden.
0x8	8	0x98110008	ERR_UNKNOWNCMDID	Unbekannte Befehl-ID.
0x9	9	0x98110009	ERR_BADTASKID	Ungültige Task-ID.
0xA	10	0x9811000A	ERR_NOIO	Kein IO.
0xB	11	0x9811000B	ERR_UNKNOWNAMSCMD	Unbekannter AMS-Befehl.
0xC	12	0x9811000C	ERR_WIN32ERROR	Win32 Fehler.
0xD	13	0x9811000D	ERR_PORTNOTCONNECTED	Port nicht verbunden.
0xE	14	0x9811000E	ERR_INVALIDAMSLENGTH	Ungültige AMS-Länge.
0xF	15	0x9811000F	ERR_INVALIDAMSNETID	Ungültige AMS Net ID.
0x10	16	0x98110010	ERR_LOWINSTLEVEL	Installations-Level ist zu niedrig –TwinCAT 2 Lizenzfehler.
0x11	17	0x98110011	ERR_NODEBUGINTAVAILABLE	Kein Debugging verfügbar.
0x12	18	0x98110012	ERR_PORTDISABLED	Port deaktiviert – TwinCAT System Service nicht gestartet.
0x13	19	0x98110013	ERR_PORTALREADYCONNECTED	Port bereits verbunden.
0x14	20	0x98110014	ERR_AMSSYNC_W32ERROR	AMS Sync Win32 Fehler.
0x15	21	0x98110015	ERR_AMSSYNC_TIMEOUT	AMS Sync Timeout.
0x16	22	0x98110016	ERR_AMSSYNC_AMSERROR	AMS Sync Fehler.
0x17	23	0x98110017	ERR_AMSSYNC_NOINDEXINMAP	Keine Index-Map für AMS Sync vorhanden.
0x18	24	0x98110018	ERR_INVALIDAMSPORT	Ungültiger AMS-Port.
0x19	25	0x98110019	ERR_NOMEMORY	Kein Speicher.
0x1A	26	0x9811001A	ERR_TCPSEND	TCP Sendefehler.
0x1B	27	0x9811001B	ERR_HOSTUNREACHABLE	Host nicht erreichbar.
0x1C	28	0x9811001C	ERR_INVALIDAMSFRAGMENT	Ungültiges AMS Fragment.
0x1D	29	0x9811001D	ERR_TLSSEND	TLS Sendefehler – Secure ADS Verbindung fehlgeschlagen.
0x1E	30	0x9811001E	ERR_ACCESSDENIED	Zugriff Verweigert – Secure ADS Zugriff verweigert.

Router Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x500	1280	0x98110500	ROUTERERR_NOLOCKEDMEMORY	Lockierter Speicher kann nicht zugewiesen werden.
0x501	1281	0x98110501	ROUTERERR_RESIZEMEMORY	Die Größe des Routerspeichers konnte nicht geändert werden.
0x502	1282	0x98110502	ROUTERERR_MAILBOXFULL	Das Postfach hat die maximale Anzahl der möglichen Meldungen erreicht.
0x503	1283	0x98110503	ROUTERERR_DEBUGBOXFULL	Das Debug Postfach hat die maximale Anzahl der möglichen Meldungen erreicht.
0x504	1284	0x98110504	ROUTERERR_UNKNOWNPORTTYPE	Der Porttyp ist unbekannt.
0x505	1285	0x98110505	ROUTERERR_NOTINITIALIZED	Router ist nicht initialisiert.
0x506	1286	0x98110506	ROUTERERR_PORTALREADYINUSE	Die Portnummer ist bereits vergeben.
0x507	1287	0x98110507	ROUTERERR_NOTREGISTERED	Der Port ist nicht registriert.
0x508	1288	0x98110508	ROUTERERR_NOMOREQUEUES	Die maximale Portanzahl ist erreicht.
0x509	1289	0x98110509	ROUTERERR_INVALIDPORT	Der Port ist ungültig.
0x50A	1290	0x9811050A	ROUTERERR_NOTACTIVATED	Der Router ist nicht aktiv.
0x50B	1291	0x9811050B	ROUTERERR_FRAGMENTBOXFULL	Das Postfach hat die maximale Anzahl für fragmentierte Nachrichten erreicht.
0x50C	1292	0x9811050C	ROUTERERR_FRAGMENTTIMEOUT	Fragment Timeout aufgetreten.
0x50D	1293	0x9811050D	ROUTERERR_TOBEREMOVED	Port wird entfernt.

Allgemeine ADS Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x700	1792	0x98110700	ADSERR_DEVICE_ERROR	Allgemeiner Gerätefehler.
0x701	1793	0x98110701	ADSERR_DEVICE_SRVNOTSUPP	Service wird vom Server nicht unterstützt.
0x702	1794	0x98110702	ADSERR_DEVICE_INVALIDGRP	Ungültige Index-Gruppe.
0x703	1795	0x98110703	ADSERR_DEVICE_INVALIDOFFSET	Ungültiger Index-Offset.
0x704	1796	0x98110704	ADSERR_DEVICE_INVALIDACCESS	Lesen oder Schreiben nicht gestattet.
0x705	1797	0x98110705	ADSERR_DEVICE_INVALIDSIZE	Parametergröße nicht korrekt.
0x706	1798	0x98110706	ADSERR_DEVICE_INVALIDDATA	Ungültige Daten-Werte.
0x707	1799	0x98110707	ADSERR_DEVICE_NOTREADY	Gerät nicht betriebsbereit.
0x708	1800	0x98110708	ADSERR_DEVICE_BUSY	Gerät beschäftigt.
0x709	1801	0x98110709	ADSERR_DEVICE_INVALIDCONTEXT	Ungültiger Kontext vom Betriebssystem - Kann durch Verwendung von ADS Bausteinen in unterschiedlichen Tasks auftreten. Abhilfe kann die Multitasking-Syncronisation in der SPS geben.
0x70A	1802	0x9811070A	ADSERR_DEVICE_NOMEMORY	Nicht genügend Speicher.
0x70B	1803	0x9811070B	ADSERR_DEVICE_INVALIDPARM	Ungültige Parameter-Werte.
0x70C	1804	0x9811070C	ADSERR_DEVICE_NOTFOUND	Nicht gefunden (Dateien,...).
0x70D	1805	0x9811070D	ADSERR_DEVICE_SYNTAX	Syntax-Fehler in Datei oder Befehl.
0x70E	1806	0x9811070E	ADSERR_DEVICE_INCOMPATIBLE	Objekte stimmen nicht überein.
0x70F	1807	0x9811070F	ADSERR_DEVICE_EXISTS	Objekt ist bereits vorhanden.
0x710	1808	0x98110710	ADSERR_DEVICE_SYMBOLNOTFOUND	Symbol nicht gefunden.
0x711	1809	0x98110711	ADSERR_DEVICE_SYMBOLVERSIONINVALID	Symbol-Version ungültig – Kann durch einen Online-Change auftreten. Erzeuge einen neuen Handle.
0x712	1810	0x98110712	ADSERR_DEVICE_INVALIDSTATE	Gerät (Server) ist im ungültigen Zustand.
0x713	1811	0x98110713	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransMode nicht unterstützt.
0x714	1812	0x98110714	ADSERR_DEVICE_NOTIFYHNDINVALID	Notification Handle ist ungültig.
0x715	1813	0x98110715	ADSERR_DEVICE_CLIENTUNKNOWN	Notification-Client nicht registriert.
0x716	1814	0x98110716	ADSERR_DEVICE_NOMOREHDLS	Keine weiteren Handles verfügbar.
0x717	1815	0x98110717	ADSERR_DEVICE_INVALIDWATCHSIZE	Größe der Notification zu groß.
0x718	1816	0x98110718	ADSERR_DEVICE_NOTINIT	Gerät nicht initialisiert.
0x719	1817	0x98110719	ADSERR_DEVICE_TIMEOUT	Gerät hat einen Timeout.
0x71A	1818	0x9811071A	ADSERR_DEVICE_NOINTERFACE	Interface Abfrage fehlgeschlagen.
0x71B	1819	0x9811071B	ADSERR_DEVICE_INVALIDINTERFACE	Falsches Interface angefordert.
0x71C	1820	0x9811071C	ADSERR_DEVICE_INVALIDCLSID	Class-ID ist ungültig.
0x71D	1821	0x9811071D	ADSERR_DEVICE_INVALIDOBJID	Object-ID ist ungültig.
0x71E	1822	0x9811071E	ADSERR_DEVICE_PENDING	Anforderung steht aus.
0x71F	1823	0x9811071F	ADSERR_DEVICE_ABORTED	Anforderung wird abgebrochen.
0x720	1824	0x98110720	ADSERR_DEVICE_WARNING	Signal-Warnung.
0x721	1825	0x98110721	ADSERR_DEVICE_INVALIDARRAYIDX	Ungültiger Array-Index.
0x722	1826	0x98110722	ADSERR_DEVICE_SYMBOLNOTACTIVE	Symbol nicht aktiv.
0x723	1827	0x98110723	ADSERR_DEVICE_ACCESSDENIED	Zugriff verweigert.
0x724	1828	0x98110724	ADSERR_DEVICE_LICENSENOTFOUND	Fehlende Lizenz.
0x725	1829	0x98110725	ADSERR_DEVICE_LICENSEEXPIRED	Lizenz abgelaufen.
0x726	1830	0x98110726	ADSERR_DEVICE_LICENSEEXCEEDED	Lizenz überschritten.
0x727	1831	0x98110727	ADSERR_DEVICE_LICENSEINVALID	Lizenz ungültig.
0x728	1832	0x98110728	ADSERR_DEVICE_LICENSESYSTEMID	Lizenzproblem: System-ID ist ungültig.
0x729	1833	0x98110729	ADSERR_DEVICE_LICENSENOTIMELIMIT	Lizenz nicht zeitlich begrenzt.
0x72A	1834	0x9811072A	ADSERR_DEVICE_LICENSEFUTUREISSUE	Lizenzproblem: Zeitpunkt in der Zukunft.
0x72B	1835	0x9811072B	ADSERR_DEVICE_LICENSETIMETOLONG	Lizenz-Zeitraum zu lang.
0x72C	1836	0x9811072C	ADSERR_DEVICE_EXCEPTION	Exception beim Systemstart.
0x72D	1837	0x9811072D	ADSERR_DEVICE_LICENSEDUPLICATED	Lizenz-Datei zweimal gelesen.
0x72E	1838	0x9811072E	ADSERR_DEVICE_SIGNATUREINVALID	Ungültige Signatur.
0x72F	1839	0x9811072F	ADSERR_DEVICE_CERTIFICATEINVALID	Zertifikat ungültig.
0x730	1840	0x98110730	ADSERR_DEVICE_LICENSEOEMNOTFOUND	Public Key vom OEM nicht bekannt.
0x731	1841	0x98110731	ADSERR_DEVICE_LICENSERESTRICTED	Lizenz nicht gültig für diese System.ID.
0x732	1842	0x98110732	ADSERR_DEVICE_LICENSEDEMODENIED	Demo-Lizenz untersagt.
0x733	1843	0x98110733	ADSERR_DEVICE_INVALIDFNCID	Funktions-ID ungültig.
0x734	1844	0x98110734	ADSERR_DEVICE_OUTOFRANGE	Außerhalb des gültigen Bereiches.
0x735	1845	0x98110735	ADSERR_DEVICE_INVALIDALIGNMENT	Ungültiges Alignment.

Hex	Dec	HRESULT	Name	Beschreibung
0x736	1846	0x98110736	ADSERR_DEVICE_LICENSEPLATFORM	Ungültiger Plattform Level.
0x737	1847	0x98110737	ADSERR_DEVICE_FORWARD_PL	Kontext – Weiterleitung zum Passiv-Level.
0x738	1848	0x98110738	ADSERR_DEVICE_FORWARD_DL	Kontext – Weiterleitung zum Dispatch-Level.
0x739	1849	0x98110739	ADSERR_DEVICE_FORWARD_RT	Kontext – Weiterleitung zur Echtzeit.
0x740	1856	0x98110740	ADSERR_CLIENT_ERROR	Clientfehler.
0x741	1857	0x98110741	ADSERR_CLIENT_INVALIDPARM	Dienst enthält einen ungültigen Parameter.
0x742	1858	0x98110742	ADSERR_CLIENT_LISTEMPTY	Polling-Liste ist leer.
0x743	1859	0x98110743	ADSERR_CLIENT_VARUSED	Var-Verbindung bereits im Einsatz.
0x744	1860	0x98110744	ADSERR_CLIENT_DUPLINVOKEID	Die aufgerufene ID ist bereits in Benutzung.
0x745	1861	0x98110745	ADSERR_CLIENT_SYNCTIMEOUT	Timeout ist aufgetreten – Die Gegenstelle antwortet nicht im vorgegebenen ADS Timeout. Die Routeneinstellung der Gegenstelle kann falsch konfiguriert sein.
0x746	1862	0x98110746	ADSERR_CLIENT_W32ERROR	Fehler im Win32 Subsystem.
0x747	1863	0x98110747	ADSERR_CLIENT_TIMEOUTINVALID	Ungültiger Client Timeout-Wert.
0x748	1864	0x98110748	ADSERR_CLIENT_PORTNOTOPEN	Port nicht geöffnet.
0x749	1865	0x98110749	ADSERR_CLIENT_NOAMSADDR	Keine AMS Adresse.
0x750	1872	0x98110750	ADSERR_CLIENT_SYNCINTERNAL	Interner Fehler in Ads-Sync.
0x751	1873	0x98110751	ADSERR_CLIENT_ADDHASH	Überlauf der Hash-Tabelle.
0x752	1874	0x98110752	ADSERR_CLIENT_REMOVEHASH	Schlüssel in der Tabelle nicht gefunden.
0x753	1875	0x98110753	ADSERR_CLIENT_NOMORESYM	Keine Symbole im Cache.
0x754	1876	0x98110754	ADSERR_CLIENT_SYNCRESINVALID	Ungültige Antwort erhalten.
0x755	1877	0x98110755	ADSERR_CLIENT_SYNCPORTLOCKED	Sync Port ist verriegelt.

RTime Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x1000	4096	0x98111000	RTERR_INTERNAL	Interner Fehler im Echtzeit-System.
0x1001	4097	0x98111001	RTERR_BADTIMERPERIODS	Timer-Wert nicht gültig.
0x1002	4098	0x98111002	RTERR_INVALIDTASKPTR	Task-Pointer hat den ungültigen Wert 0 (null).
0x1003	4099	0x98111003	RTERR_INVALIDSTACKPTR	Stack-Pointer hat den ungültigen Wert 0 (null).
0x1004	4100	0x98111004	RTERR_PRIOEXISTS	Die Request Task Priority ist bereits vergeben.
0x1005	4101	0x98111005	RTERR_NOMORETCB	Kein freier TCB (Task Control Block) verfügbar. Maximale Anzahl von TCBs beträgt 64.
0x1006	4102	0x98111006	RTERR_NOMORESEMAS	Keine freien Semaphoren zur Verfügung. Maximale Anzahl der Semaphoren beträgt 64.
0x1007	4103	0x98111007	RTERR_NOMOREQUEUES	Kein freier Platz in der Warteschlange zur Verfügung. Maximale Anzahl der Plätze in der Warteschlange beträgt 64.
0x100D	4109	0x9811100D	RTERR_EXTIRQALREADYDEF	Ein externer Synchronisations-Interrupt wird bereits angewandt.
0x100E	4110	0x9811100E	RTERR_EXTIRQNOTDEF	Kein externer Sync-Interrupt angewandt.
0x100F	4111	0x9811100F	RTERR_EXTIRQINSTALLFAILED	Anwendung des externen Synchronisierungs-Interrupts ist fehlgeschlagen.
0x1010	4112	0x98111010	RTERR_IRQLNOTLESSOREQUAL	Aufruf einer Service-Funktion im falschen Kontext
0x1017	4119	0x98111017	RTERR_VMXNOTSUPPORTED	Intel VT-x Erweiterung wird nicht unterstützt.
0x1018	4120	0x98111018	RTERR_VMXDISABLED	Intel VT-x Erweiterung ist nicht aktiviert im BIOS.
0x1019	4121	0x98111019	RTERR_VMXCONTROLSMISSING	Fehlende Funktion in Intel VT-x Erweiterung.
0x101A	4122	0x9811101A	RTERR_VMXENABLEFAILS	Aktivieren von Intel VT-x schlägt fehl.

Spezifische positive HRESULT Return Codes:

HRESULT	Name	Beschreibung
0x0000_0000	S_OK	Kein Fehler.
0x0000_0001	S_FALSE	Kein Fehler. Bsp.: erfolgreiche Abarbeitung, bei der jedoch ein negatives oder unvollständiges Ergebnis erzielt wurde.
0x0000_0203	S_PENDING	Kein Fehler. Bsp.: erfolgreiche Abarbeitung, bei der jedoch noch kein Ergebnis vorliegt.
0x0000_0256	S_WATCHDOG_TIMEOUT	Kein Fehler. Bsp.: erfolgreiche Abarbeitung, bei der jedoch eine Zeitüberschreitung eintrat.

TCP Winsock-Fehlercodes

Hex	Dec	Name	Beschreibung
0x274C	10060	WSAETIMEDOUT	Verbindungs Timeout aufgetreten - Fehler beim Herstellen der Verbindung, da die Gegenstelle nach einer bestimmten Zeitspanne nicht ordnungsgemäß reagiert hat, oder die hergestellte Verbindung konnte nicht aufrecht erhalten werden, da der verbundene Host nicht reagiert hat.
0x274D	10061	WSAECONNREFUSED	Verbindung abgelehnt - Es konnte keine Verbindung hergestellt werden, da der Zielcomputer dies explizit abgelehnt hat. Dieser Fehler resultiert normalerweise aus dem Versuch, eine Verbindung mit einem Dienst herzustellen, der auf dem fremden Host inaktiv ist—das heißt, einem Dienst, für den keine Serveranwendung ausgeführt wird.
0x2751	10065	WSAEHOSTUNREACH	Keine Route zum Host - Ein Socketvorgang bezog sich auf einen nicht verfügbaren Host.
Weitere Winsock-Fehlercodes: Win32-Fehlercodes			

7.2 Aufzählungen

7.2.1 ETcIotHttpRequestError

```
TYPE ETcIotHttpRequestError:
(
    HTTP_REQ_ERR_BUSY:=-1,
    HTTP_REQ_ERR_SUCCESS:=0,
    HTTP_REQ_ERR_NOMEM:=1,
    HTTP_REQ_ERR_CREATE_PROTOCOL:=2,
    HTTP_REQ_ERR_CONN_INVAL:=3,
    HTTP_REQ_ERR_NO_CONN:=4,
    HTTP_REQ_ERR_CONN_REFUSED:=5,
    HTTP_REQ_ERR_NOT_FOUND:=6,
    HTTP_REQ_ERR_CONN_LOST:=7,
    HTTP_REQ_ERR_TLS:=8,
    HTTP_REQ_ERR_NOT_SUPPORTED:=10,
    HTTP_REQ_ERR_AUTH:=11,
    HTTP_REQ_ERR_ACL_DENIED:=12,
    HTTP_REQ_ERR_UNKNOWN:=13,
    HTTP_REQ_ERR_ERRNO:=14,
    HTTP_REQ_ERR_EAI:=15,
    HTTP_REQ_ERR_PROXY:=16,
    HTTP_REQ_ERR_TLS_CA_NOTFOUND:=17,
    HTTP_REQ_ERR_TLS_CERT_NOTFOUND:=18,
    HTTP_REQ_ERR_TLS_KEY_NOTFOUND:=19,
    HTTP_REQ_ERR_TLS_CA_INVALID:=20,
    HTTP_REQ_ERR_TLS_CERT_INVALID:=21,
    HTTP_REQ_ERR_TLS_KEY_INVALID:=22,
    HTTP_REQ_ERR_TLS_VERIFY_FAIL:=23,
    HTTP_REQ_ERR_TLS_SETUP:=24,
    HTTP_REQ_ERR_TLS_HANDSHAKE_FAIL:=25,
    HTTP_REQ_ERR_TLS_CIPHER_INVALID:=26,
    HTTP_REQ_ERR_TLS_VERSION_INVALID:=27,
    HTTP_REQ_ERR_TLS_PSK_INVALID:=28,
    HTTP_REQ_ERR_TLS_CRL_NOTFOUND:=29,
    HTTP_REQ_ERR_TLS_CRL_INVALID:=30,
    HTTP_REQ_ERR_FINALIZE_DISCONNECT:=31,
    HTTP_REQ_ERR_BIND:=32,
    HTTP_REQ_ERR_BIND_ADDR_INUSE:=33,
    HTTP_REQ_ERR_BIND_ADDR_INVAL:=34,
    HTTP_REQ_ERR_CREATE:=35,
    HTTP_REQ_ERR_CREATE_TYPE:=36,
    HTTP_REQ_ERR_CONN:=37,
    HTTP_REQ_ERR_CONN_TIMEDOUT:=38,
    HTTP_REQ_ERR_CONN_HOSTUNREACH:=39,
    HTTP_REQ_ERR_TLS_CERT_EXPIRED:=40,
    HTTP_REQ_ERR_TLS_CN_MISMATCH:=41,
    HTTP_REQ_ERR_INV_PARAM:=1000,
    HTTP_REQ_ERR_FIFO_FULL:=1001,
    HTTP_REQ_ERR_TCP_SEND:=1002,
    HTTP_REQ_ERR_CANCELLED:=1003,
    HTTP_REQ_ERR_RESPONSE_TIMEDOUT:=1004,
    HTTP_REQ_ERR_INV_HDR_SIZE:=1005,
    HTTP_REQ_ERR_INV_ENCODING:=1006,
    HTTP_REQ_ERR_INV_CONTENT_SIZE:=1007,
    HTTP_REQ_ERR_INV_CHUNK_SIZE:=1008,
```

```
    HTTP_REQ_ERR_PARSE_HDR:=1100,
    HTTP_REQ_ERR_PARSE_HDR_FIELD:=1101,
    HTTP_REQ_ERR_PARSE_HDR_FIELD_NAME:=1102,
    HTTP_REQ_ERR_PARSE_HDR_FIELD_VAL:=1103,
    HTTP_REQ_ERR_PARSE_STATUS_LINE:=1104,
    HTTP_REQ_ERR_PARSE_CHUNK:=1105,
    HTTP_REQ_ERR_PARSE_CHUNK_SIZE:=1106,
    HTTP_REQ_ERR_PARSE_CHUNK_EXT_NAME:=1107,
    HTTP_REQ_ERR_PARSE_CHUNK_EXT_VAL:=1108,
    HTTP_REQ_ERR_PARSE_CHUNK_DATA:=1109,
    HTTP_REQ_ERR_PARSE_CHUNK_TRAILER:=1110
) DINT;
END_TYPE
```

7.2.2 ETcIotHttpRequestType

```
TYPE ETcIotHttpRequestType:
(
    HTTP_GET:=0,
    HTTP_POST:=1,
    HTTP_PUT:=2,
    HTTP_DELETE:=3,
    HTTP_TRACE:=4,
    HTTP_OPTIONS:=5,
    HTTP_PATCH:=6,
    HTTP_CONNECT:=7,
    HTTP_HEAD:=8
) DINT;
END_TYPE
```

7.2.3 E_IotHttpCompressionMode

```
TYPE E_IotHttpCompressionMode:
(
    NoCompression:=0,
    Deflate:=1,
    Gzip:=2
) UDINT;
END_TYPE
```

7.3 Verwendete Cipher-Suites

Der TwinCAT IoT-Treiber unterstützt sichere Übertragung unter Verwendung des TLS-Standards in der Version 1.2. Bis zu dieser Version ist eine Cipher-Suite eine Zusammensetzung von Algorithmen (Schlüsselaustausch, Authentifizierung, Verschlüsselung, MAC) zur sicheren Übertragung.

Nachfolgend sind alle aktuell vom IoT-Treiber unterstützten Cipher-Suites aufgeführt. Der Informationsstand bezieht sich hier auf die TwinCAT-Version 3.1.4024.12.

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

Mehr Informationen:
www.beckhoff.de/tf6760/

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Deutschland
Telefon: +49 5246 9630
info@beckhoff.de
www.beckhoff.de