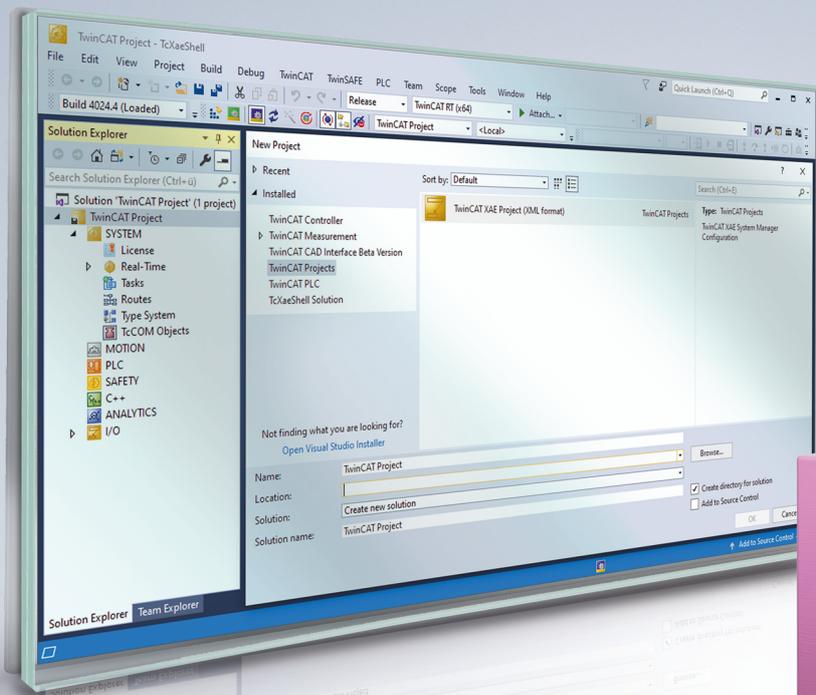


BECKHOFF New Automation Technology

Handbuch | DE

TF6760

TwinCAT 3 | IoT HTTPS/REST



Inhaltsverzeichnis

1	Vorwort	5
1.1	Hinweise zur Dokumentation	5
1.2	Zu Ihrer Sicherheit	6
1.3	Hinweise zur Informationssicherheit	7
2	Übersicht	8
3	Installation	9
3.1	Systemanforderungen	9
3.2	Installation	9
3.3	Lizenzierung	9
4	Technische Einführung	12
4.1	HTTP/HTTPS	12
4.2	REST-API	13
4.3	Unterstützte Funktionen	14
4.4	Technische Einschränkungen	15
4.5	Kompression	15
4.6	Sicherheit	16
4.6.1	Transportebene	16
4.6.2	Applikationsebene	23
4.7	Neuparametrierung	24
4.8	URL-Redirects	24
4.9	Cookies	25
5	SPS API	26
5.1	Tc3_lotBase	26
5.1.1	Funktionsbausteine	26
5.1.2	Datentypen	39
5.1.3	Parameterliste	41
5.2	Tc3_JsonXml	42
5.2.1	Funktionsbausteine	42
5.2.2	Schnittstellen	233
6	Beispiele	239
6.1	lotHttpSamples	239
6.1.1	OpenWeatherMap	240
6.1.2	PostmanEcho	240
6.1.3	AWS IoT	240
6.1.4	Philips Hue	240
6.1.5	Telegram	241
6.1.6	AWS Service Configuration	244
7	Anhang	245
7.1	ADS Return Codes	245

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, stets die aktuell gültige 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 weiterentwickelt.

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®, ATRO®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, MX-System®, Safety over EtherCAT®, TC/BSD®, TwinCAT®, TwinCAT/BSD®, TwinSAFE®, XFC®, XPlanar® und XTS® 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 Kennzeichnungen führen.



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

Copyright

© Beckhoff Automation GmbH & Co. KG, Deutschland.

Weitergabe sowie Vervielfältigung dieses Dokuments, Verwertung und Mitteilung seines Inhalts sind verboten, soweit nicht ausdrücklich gestattet.

Zuwendungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patent-, Gebrauchsmuster- oder Geschmacksmustereintragung vorbehalten.

Fremdmarken

In dieser Dokumentation können Marken Dritter verwendet werden. Die zugehörigen Markenvermerke finden Sie unter: <https://www.beckhoff.com/trademarks>.

1.2 Zu Ihrer Sicherheit

Sicherheitsbestimmungen

Lesen Sie die folgenden Erklärungen zu Ihrer Sicherheit.
Beachten und befolgen Sie stets produktspezifische Sicherheitshinweise, die Sie gegebenenfalls an den entsprechenden Stellen in diesem Dokument vorfinden.

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.

Signalwörter

Im Folgenden werden die Signalwörter eingeordnet, die in der Dokumentation verwendet werden. Um Personen- und Sachschäden zu vermeiden, lesen und befolgen Sie die Sicherheits- und Warnhinweise.

Warnungen vor Personenschäden

GEFAHR

Es besteht eine Gefährdung mit hohem Risikograd, die den Tod oder eine schwere Verletzung zur Folge hat.

WARNUNG

Es besteht eine Gefährdung mit mittlerem Risikograd, die den Tod oder eine schwere Verletzung zur Folge haben kann.

VORSICHT

Es besteht eine Gefährdung mit geringem Risikograd, die eine mittelschwere oder leichte Verletzung zur Folge haben kann.

Warnung vor Umwelt- oder Sachschäden

HINWEIS

Es besteht eine mögliche Schädigung für Umwelt, Geräte oder Daten.

Information zum Umgang mit dem Produkt



Diese Information beinhaltet z. B.:
Handlungsempfehlungen, Hilfestellungen oder weiterführende Informationen zum Produkt.

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.

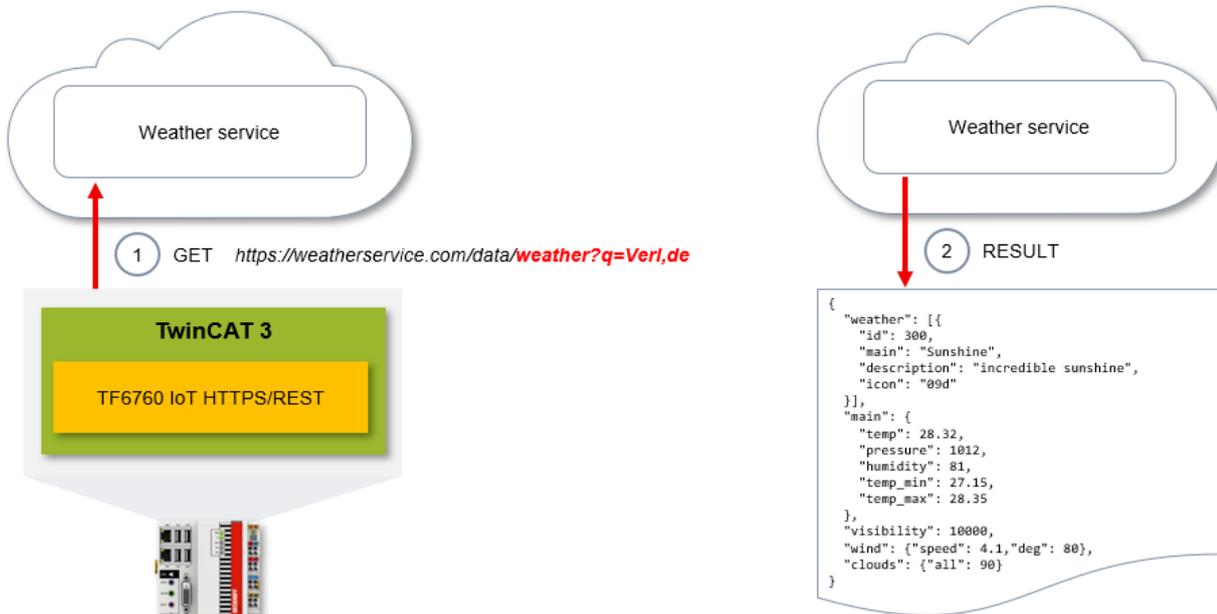
Zusätzlich sollten die Empfehlungen von Beckhoff zu entsprechenden Schutzmaßnahmen beachtet werden. Weiterführende Informationen über Informationssicherheit und Industrial Security finden Sie in unserem <https://www.beckhoff.de/secguide>.

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.



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, TwinCAT/BSD
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_lotBase)
- 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](#)“.

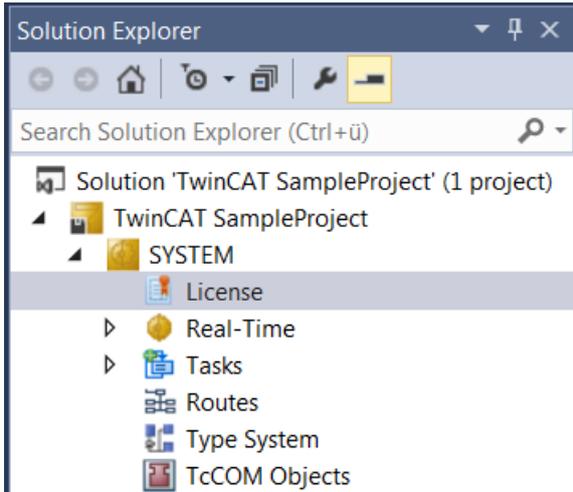
Lizenzierung der 7-Tage-Testversion einer TwinCAT 3 Function



Eine 7-Tage-Testversion kann nicht für einen [TwinCAT-3-Lizenz-Dongle](#) freigeschaltet werden.

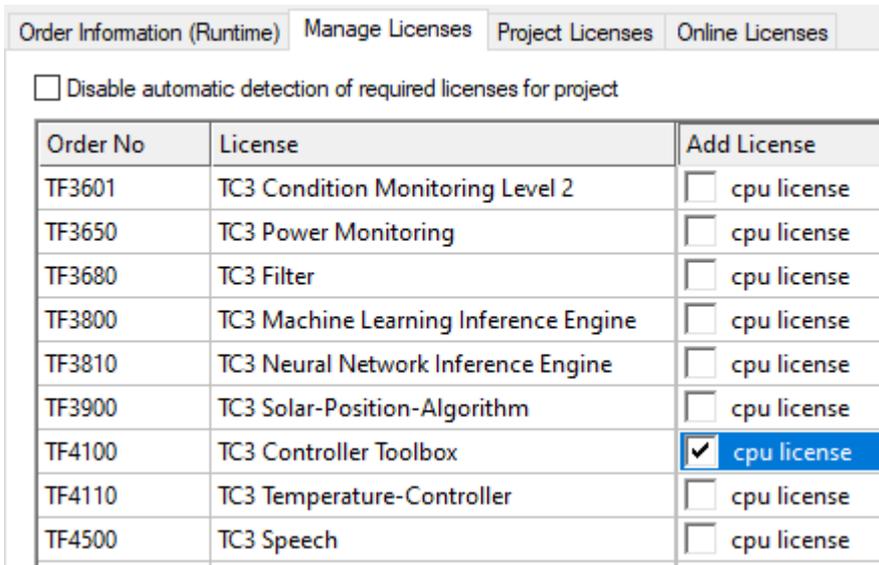
1. Starten Sie die TwinCAT-3-Entwicklungsumgebung (XAE).
2. Öffnen Sie ein bestehendes TwinCAT-3-Projekt oder legen Sie ein neues Projekt an.
3. Wenn Sie die Lizenz für ein Remote-Gerät aktivieren wollen, stellen Sie das gewünschte Zielsystem ein. Wählen Sie dazu in der Symbolleiste in der Drop-down-Liste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

The screenshot shows a software interface with several sections:

- Order Information (Runtime)**: Includes tabs for 'Manage Licenses', 'Project Licenses', and 'Online Licenses'. Below are fields for 'License Device' (set to 'Target (Hardware Id)'), 'System Id' (containing '2DB25408-B4CD-81DF-5488-6A3D9B49EF19'), and 'Platform' (set to 'other (91)').
- License Request**: Includes a 'Provider' dropdown set to 'Beckhoff Automation', a 'Generate File...' button, and input fields for 'License Id', 'Customer Id', and 'Comment'.
- License Activation**: Contains two buttons: '7 Days Trial License...' (highlighted with a red box) and 'License Response File...'.

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

The dialog box is titled 'Enter Security Code' and contains the following elements:

- Text: 'Please type the following 5 characters:'
- Code display: 'Kg8T4' (highlighted with a red box)
- Input field: A two-character input field with a red border and a blue outline.
- Buttons: 'OK' (highlighted with a red box) and 'Cancel'.

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.



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 Unterstützte Funktionen

Die folgende Tabelle gibt einen Überblick über alle unterstützten Funktionen. Grundsätzlich wird HTTP in Version 1.1 unterstützt.

Funktion	Min. TwinCAT-Version	Beschreibung
HTTP-Requests	3.1 Build 4024.7	GET, POST, PUT, CONNECT, DELETE, HEAD, OPTIONS, TRACE, PATCH.
Cipher Suites	3.1 Build 4024.7	Angabe der zu verwendenden Cipher Suites.
Datenformat	3.1 Build 4024.7	Das Datenformat kann frei definiert werden. Entsprechende Hilfsbibliotheken zur einfachen Verwendung von JSON und XML stehen zur Verfügung.
Header-Fields	3.1 Build 4024.7	Header-Fields können frei definiert oder angepasst werden.
Kompression	3.1 Build 4024.11	Es werden Kompressionsalgorithmen [► 15] verwendet, um die Datenmenge von Nachrichten zu verringern. Unterstützt werden in Send- und Empfangsrichtung gzip und deflate.
Persistent Connections (Keep-Alive)	3.1 Build 4024.7	Die Verbindung zu einem HTTP-Server wird aufrechterhalten, anstatt dass sie nach einem Request wieder abgebaut wird.
Pre-Shared Key (PSK)	3.1 Build 4024.7	Ein PSK kann zur Absicherung des Transportkanals (mittels TLS) verwendet werden.
Transport Layer Security (TLS) – Version 1.1 und 1.2	3.1 Build 4024.7	TLS kann zur Absicherung des Transportkanals verwendet werden.
Transport Layer Security (TLS) – Version 1.1 und 1.3	3.1 Build 4026.0	TLS kann zur Absicherung des Transportkanals verwendet werden.
Zertifikate	3.1 Build 4024.7	Zertifikate können zur Absicherung des Transportkanals (mittels TLS) verwendet werden. Hierbei wird das PEM-Format verwendet.
Zertifikatssperlisten (CRL)	3.1 Build 4024.7	Verwendung von Zertifikatssperlisten.

4.4 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 <code>FB_lotHttpClient</code> [▶ 29] 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 Datenmengen	Bei der Kommunikation von großen Datenmengen sollte sich der Anwender bewusst sein, dass dies zu einer Belastung der Echtzeit führt, da diese große Menge an Daten kopiert werden muss. Außerdem wird Router-Speicher verwendet.

4.5 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_lotBase`-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_lotHttpRequest` [▶ 32]-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](#) [▶ 240]). Der TwinCAT HTTP-Client teilt der REST API per Header-Field mit, dass sowohl deflate als auch gzip in Empfangsrichtung unterstützt werden.

```

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

4.6 Sicherheit

Bei der Betrachtung der Absicherung einer Datenkommunikation lassen sich zwei Ebenen voneinander unterscheiden. Zum einen die Absicherung des [Transportkanals](#) [[16](#)] und zum anderen die Absicherung auf Applikationsebene.

4.6.1 Transportebene

Für die sichere Übertragung von Daten wird im TwinCAT IoT-Treiber der weltweit gängige Standard Transport Layer Security (TLS) verwendet. Das folgende Kapitel beschreibt den TLS-Kommunikationsablauf exemplarisch am Beispiel von TLS-Version 1.2.

TLS ist ein Standard, welcher eine Kombination aus symmetrischer und asymmetrischer Kryptographie darstellt und versendete Daten vor unbefugtem Zugriff und Manipulation durch Dritte schützt. Außerdem wird die Authentifizierung der Kommunikationsteilnehmer zur gegenseitigen Identitätsprüfung von TLS unterstützt.

● Inhalt dieses Kapitels

I 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.

Unterstützte Funktionen

Der TwinCAT IoT -Treiber ermöglicht die Verwendung der folgenden TLS-Funktionen.

Funktion	Beschreibung
Selbst signierte Clientzertifikate	Verwendung eines selbst-signierten Clientzertifikats zur Authentifizierung am Message Broker.
CA-signierte Clientzertifikate	Verwendung eines CA-signierten Clientzertifikats zur Authentifizierung am Message Broker. Das CA-Zertifikat kann zum Herstellen einer Vertrauensstellung ebenfalls angegeben werden.
Zertifikatssperrlisten	Verwendung von Zertifikatssperrlisten (englisch: Certificate Revocation Lists (CRL)).
Pre-Shared Key (PSK)	Verwendung eines Pre-Shared Key (PSK) zur Authentifizierung am Message Broker.

Definition Cipher-Suite

Eine Cipher Suite ist per Definition 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.

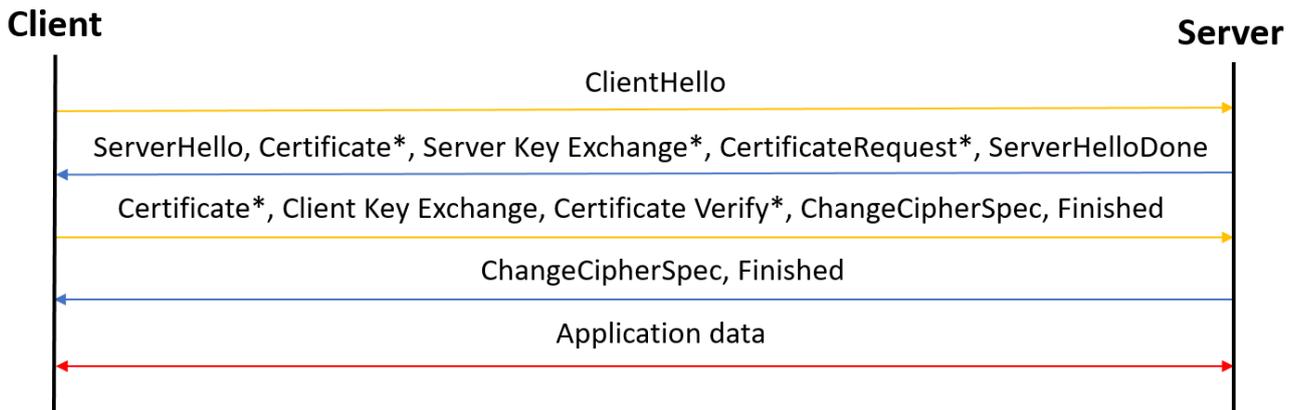
TLS-Kommunikationsablauf

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](#) [[19](#)])
- Client und Server weisen sich per Zertifikat aus (siehe [Client/Server-Zertifikat](#) [[20](#)])
- Pre-Shared-Keys (siehe [Pre-Shared-Keys](#) [[21](#)])

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



● Beispielhafte Erläuterung anhand von RSA

i 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.

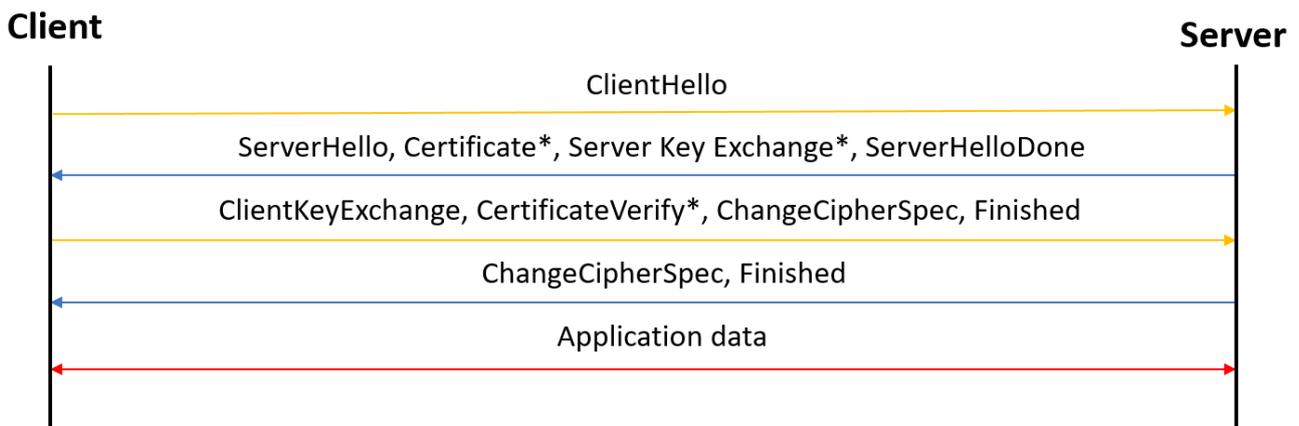
Die folgende Tabelle erläutert die einzelnen Schritte aus dem oben dargestellten Kommunikationsablauf.

Schritt	Beschreibung
ClientHello	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.
ServerHello	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.
ServerKeyExchange	Bei bestimmten Schlüsselaustauschalgorithmien 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.
CertificateRequest	Der Server fordert vom Client ein Zertifikat an, um die Identität des Clients verifizieren zu können.
ServerHelloDone	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.
ClientKeyExchange	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.
CertificateVerify	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“.
ChangeCipherSpec	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.

Schritt	Beschreibung
ChangeCipherSpec	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.
ApplicationData	Client und Server kommunizieren nach Abschluss des TLS-Verbindungsaufbaus mit symmetrischer Kryptographie.

4.6.1.1 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 \[► 16\]](#) 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 auf den HTTP-Client, den MQTT-Client und den WebSocket-Client, exemplarisch wird der HTTP-Client verwendet.

```
PROGRAM MAIN
VAR
    fbClient : FB_IotHttpClient;
END_VAR

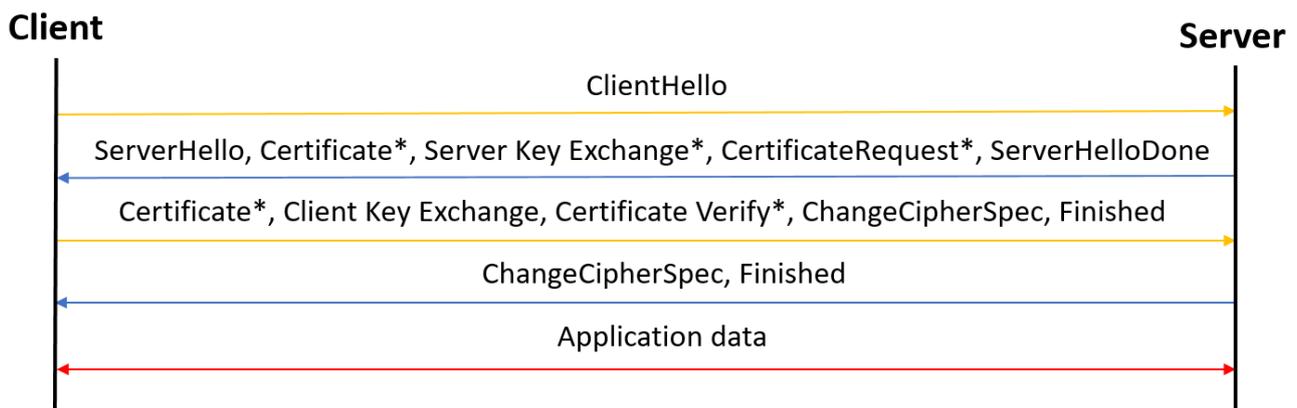
fbClient.stTLS.sCA:= 'C:\TwinCAT\3.1\Config\Certificates\someCA.pem';
```

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;
```

4.6.1.2 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](#) [► 19]-Kapitel leicht abgeänderte Kommunikationsablauf wird in der folgenden Grafik visualisiert. Die einzelnen Schritte des TLS-Verbindungsaufbaus sind im Kapitel [Transportebene](#) [► 16] 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 auf den HTTP-Client, den MQTT-Client und den WebSocket-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';
```

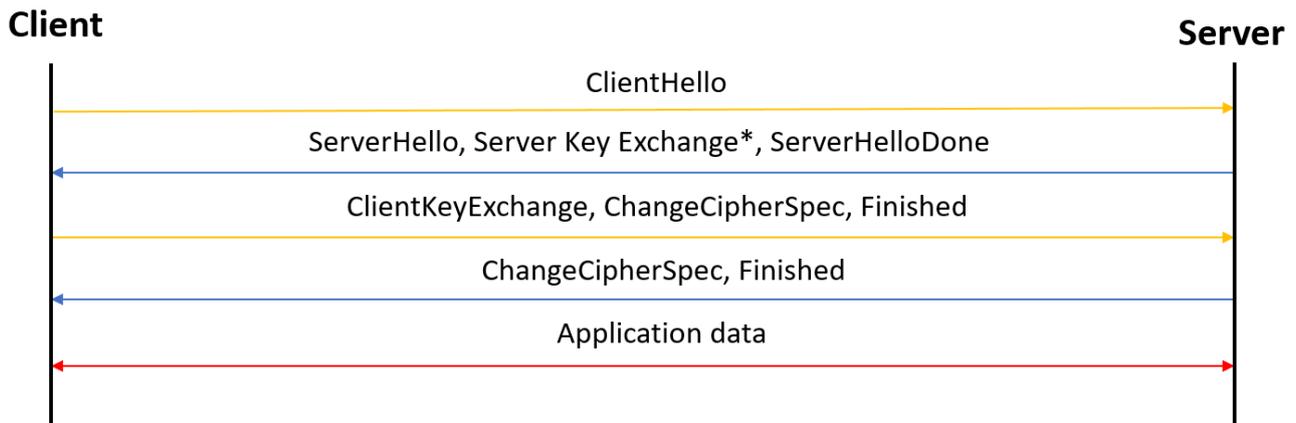
In dem Fall, dass ein Client-Zertifikat gesetzt ist, muss auch ein CA-Zertifikat zur Validierung des Server-Zertifikats gesetzt werden. Das ist durch das Verhalten des im IoT-Treiber verwendeten Security-Frameworks bedingt.

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.

4.6.1.3 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 auf den HTTP-Client, den MQTT-Client und den WebSocket-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;
    
```

4.6.1.4 Unterstützte Cipher-Suites

Der TwinCAT IoT-Treiber unterstützt eine sichere Datenübertragung unter Verwendung des TLS-Standards. Nachfolgend finden Sie eine Übersicht zu allen vom IoT-Treiber unterstützten Cipher-Suites, abhängig von der TwinCAT Version.

TwinCAT 3.1 Build 4024.x

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

TwinCAT 3.1 Build 4026.x

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

4.6.2 Applikationsebene

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

4.6.2.1 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.

4.6.2.2 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.

4.7 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_lotHttpClient [► 29]). 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.

4.8 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.

4.9 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 SPS API

5.1 Tc3_lotBase

5.1.1 Funktionsbausteine

5.1.1.1 FB_IotHttpAwsSigV4HeaderFieldMap



Dieser Funktionsbaustein ermöglicht das Signieren eines Headers mit AWS Signature Version 4 (siehe [AWS Signature Version 4](#) [► 24]).

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.

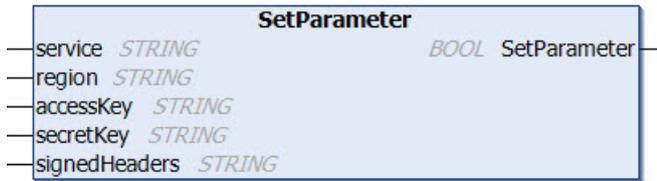
🔗 Methoden

Name	Beschreibung
AddField [► 27]	Fügt einer Instanz dieses Funktionsbausteins ein Header-Feld hinzu.
SetParameter [► 27]	Setzt die benötigten Parameter an einem für Anfragen an AWS-Services gedachten Header und signiert diesen.

Voraussetzungen

Entwicklungsumgebung	Zielformat	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4024.12 oder höher	IPC oder CX (x86, x64, Arm®)	Tc3_lotBase

5.1.1.1.1 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
```

📌 Rückgabewert

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

📌 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.

5.1.1.1.2 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
```

Rückgabewert

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

Eingänge

Name	Typ	Beschreibung
bAppendValue	BOOL	Ersetzt den vorhandenen Header-Feldwert (FALSE) oder hängt den Wert an den vorhandenen Wert an (TRUE).

Ein-/Ausgänge

Name	Typ	Beschreibung
sField	STRING	Name des Feldes, das dem Header der HTTP-Anfrage hinzugefügt wird.
sValue	STRING	Wert des hinzugefügten Header-Feldes.

5.1.1.1.3 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
```

Rückgabewert

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

Eingänge

Name	Typ	Beschreibung
pValue	PVOID	Zeiger auf den Datenspeicher, wo das gefundene Header-Field hing gespeichert werden soll.
nValueSize	UDINT	Die Größe der Speichervariablen.

Ein-/Ausgänge

Name	Typ	Beschreibung
sField	STRING	Name des Feldes, das in dem Header gefunden werden soll.

5.1.1.2 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 [▶ 39]	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.

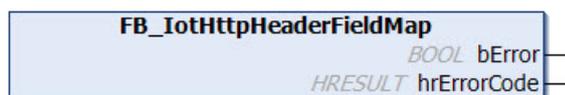
Methoden

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. Wenn die Methode das erste Mal aufgerufen wird, bevor die Parameter geändert werden, werden Default-Werte verwendet. Dann kann eine Änderung der Parameter nur über die Neuparametrierung [► 24] erreicht werden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4024.7 oder höher	IPC oder CX (x86, x64, Arm®)	Tc3_lotBase

5.1.1.3 FB_IotHttpRequestFieldMap



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_IotHttpRequestFieldMap
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 [▶ 31]	Fügt einer Instanz dieses Funktionsbausteins ein Header-Feld hinzu.
FindField [▶ 32]	Sucht in einer Instanz des Funktionsbausteins FB_lotHttpHeaderFieldMap nach einem bestimmten Header-Feld.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4024.7 oder höher	IPC oder CX (x86, x64, Arm®)	Tc3_lotBase

5.1.1.3.1 AddField



Diese Methode fügt einer Instanz des Funktionsbausteins FB_lotHttpHeaderFieldMap 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
```

 **Rückgabewert**

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

 **Eingänge**

Name	Typ	Beschreibung
bAppendValue	BOOL	Ersetzt den vorhandenen Header-Feldwert (FALSE) oder hängt den Wert an den vorhandenen Wert an (TRUE).

Ein-/Ausgänge

Name	Typ	Beschreibung
sField	STRING	Name des Feldes, das dem Header der HTTP-Anfrage hinzugefügt wird.
sValue	STRING	Wert des hinzugefügten Header-Feldes.

5.1.1.3.2 FindField



Diese Methode sucht in einer Instanz des Funktionsbausteins FB_IotHttpRequest nach einem bestimmten Header-Feld.

Syntax

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

Rückgabewert

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

Eingänge

Name	Typ	Beschreibung
pValue	PVOID	Zeiger auf den Datenspeicher, wo das gefundene Header-Feld hingespeichert werden soll.
nValueSize	UDINT	Die Größe der Speichervariablen.

Ein-/Ausgänge

Name	Typ	Beschreibung
sField	STRING	Name des Feldes, das in dem Header gefunden werden soll.

5.1.1.4 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_IotHttpRequestCompressionMode := 'E_IotHttpRequestCompressionMode
.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_IotHttpRequestCompressionMode	Aufzählung der verschiedenen Kompressions-Modi. Weitere Informationen werden im Kapitel Kompression 15] 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 [▶ 34]	Liest die Antwort vom HTTP-Server.
GetHeaderField [▶ 35]	Liest den Wert eines bestimmten Header-Feldes.
GetJsonDomContent [▶ 35]	Parst die Serverantwort als JSON-Objekt.
SendJsonDomRequest [▶ 36]	Sendet eine Anfrage an den Server als JSON-Objekt.
SendRequest [▶ 37]	Sendet einen HTTP-Befehl an den Server.
SendXmlDomRequest [▶ 38]	Sendet eine Anfrage an den Server als XML-Objekt.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4024.7 oder höher	IPC oder CX (x86, x64, Arm®)	Tc3_lotBase

5.1.1.4.1 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_lotHttpRequest` `FALSE` zurückgeben.

Syntax

```

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

Rückgabewert

Name	Typ	Beschreibung
<code>GetContent</code>	<code>BOOL</code>	Gibt <code>TRUE</code> zurück, wenn der Methodenaufruf erfolgreich war.

Eingänge

Name	Typ	Beschreibung
<code>pContent</code>	<code>PVOID</code>	Zeiger auf den Datenspeicher, wo die gelesenen Daten gespeichert werden sollen.
<code>nContentSize</code>	<code>UDINT</code>	Größe der empfangenen Nachricht.
<code>bSetNullTermination</code>	<code>BOOL</code>	Wenn die empfangene Nachricht ein <code>STRING</code> ist, kann sie entweder mit oder ohne Nullterminierung spezifiziert sein. Wenn <code>bSetNullTermination</code> auf <code>TRUE</code> gesetzt wird, wird eine Nullterminierung erzwungen. Hier ist zu beachten, dass <code>nContentSize</code> dann um <code>+1</code> erhöht werden muss.

5.1.1.4.2 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_lotHttpRequest FALSE zurückgeben.

Syntax

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

Rückgabewert

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

Eingänge

Name	Typ	Beschreibung
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.

Ein-/Ausgänge

Name	Typ	Beschreibung
sField	STRING	Name des Feldes, das dem Header der HTTP-Anfrage hinzugefügt wird.

5.1.1.4.3 GetJsonDomContent



Unter Verwendung von [GetContent](#) [▶ 34] wird die Antwort eines HTTP-Servers in eine Variable gespeichert. Wenn es sich bei der empfangenen Nachricht um einen JSON-String handelt, kann dieser im Anschluss in ein JSON-Objekt gespeichert werden, um die Funktionen der SPS-Bibliothek [Tc3_JsonXml](#) [▶ 42] nutzen zu können.

Die Methode GetJsonDomContent speichert die Antwort direkt in ein JSON-Objekt. Somit wird der Schritt der String-Konvertierung übersprungen. Um den Inhalt einer HTTP-Antwort auslesen zu können, muss der Ausgang bBusy des Funktionsbausteins FB_lotHttpRequest FALSE zurückgeben.

Syntax

```

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

```

📌 Rückgabewert

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

📌 Eingänge

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

5.1.1.4.4 SendJsonDomRequest

SendJsonDomRequest		BOOL SendJsonDomRequest
sUri	STRING	
fbClient	REFERENCE TO FB_IotHttpClient	
eRequestType	ETcIotHttpRequestType	
fbJson	REFERENCE TO FB_JsonDomParserBase	
fbHeader	REFERENCE TO FB_IotHttpHeaderFieldMap	

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

```

📌 Rückgabewert

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

 **Eingänge**

Name	Typ	Beschreibung
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_IotHttpRequestHeaderFieldMap	Die Instanz von IotHttpRequestHeaderFieldMap für die Bearbeitung des Headers der HTTP-Anfrage.

 **Ein-/Ausgänge**

Name	Typ	Beschreibung
sUri	STRING	Die Kennung der Ressource auf dem Host. Beachten Sie, dass die sUri an den Hostnamen des IotHttpClient angehängt wird.

5.1.1.4.5 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_IotHttpRequestHeaderFieldMap;
END_VAR
```

 **Rückgabewert**

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

Eingänge

Name	Typ	Beschreibung
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.

Ein-/Ausgänge

Name	Typ	Beschreibung
sUri	STRING	Die Kennung der Ressource auf dem Host. Beachten Sie, dass die sUri an den Hostnamen des IotHttpClient angehängt wird.

5.1.1.4.6 SendXmlDomRequest



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

Syntax

```
METHOD SendXmlDomRequest : BOOL
VAR_IN_OUT CONSTANT
    sUri      : STRING;
END_VAR
VAR_INPUT
    fbClient   : REFERENCE TO FB_IotHttpClient;
    eRequestType : ETcIotHttpRequestType;
    fbXml      : REFERENCE TO FB_XmlDomParser;
    fbHeader   : REFERENCE TO FB_IotHttpHeaderFieldMap;
END_VAR
```

Rückgabewert

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

 **Eingänge**

Name	Typ	Beschreibung
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.
fbXml	REFERENCE TO FB_XmlDomParser	Die Instanz von XmlDomParser zum Parsen eines Strings in ein XML-Objekt.
fbHeader	REFERENCE TO FB_IotHttpHeaderFieldMap	Die Instanz von IotHttpHeaderFieldMap für die Bearbeitung des Headers der HTTP-Anfrage.

 /  **Ein-/Ausgänge**

Name	Typ	Beschreibung
sUri	STRING	Die Kennung der Ressource auf dem Host. Beachten Sie, dass die sUri an den Hostnamen des IotHttpClient angehängt wird.

5.1.2 Datentypen

5.1.2.1 ST_IotSocketTls

Der folgende Typ enthält die TLS-Sicherheitseinstellungen für den HTTP-Client und den WebSocket-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 Cipher Suites, angegeben im OpenSSL-String-Format
sVersion	STRING(80)	Zu verwendende TLS-Version
bNoServerCertCheck	BOOL	Deaktiviert die Überprüfung des Server-Zertifikats auf Gültigkeit. Wenn ohne TLS-Verschlüsselung (HTTP/WebSockets) kommuniziert werden soll, muss dieser Wert auf FALSE bleiben.
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.

5.1.2.2 ETcIotHttpRequestError

```

TYPE ETcIotHttpRequestError:
(
  HTTP_REQ_ERR_BUSY:=-1,
  HTTP_REQ_ERR_SUCCESS:=0,
  HTTP_REQ_ERR_NOMEM:=1,
  HTTP_REQ_ERR_CREATE_PROTOCOL:=2,
  HTTP_REQ_ERR_CONN_INVALID:=3,
  HTTP_REQ_ERR_NO_CONN:=4,
  HTTP_REQ_ERR_CONN_REFUSED:=5,
  HTTP_REQ_ERR_NOT_FOUND:=6,
  HTTP_REQ_ERR_CONN_LOST:=7,
  HTTP_REQ_ERR_TLS:=8,
  HTTP_REQ_ERR_NOT_SUPPORTED:=10,
  HTTP_REQ_ERR_AUTH:=11,
  HTTP_REQ_ERR_ACL_DENIED:=12,
  HTTP_REQ_ERR_UNKNOWN:=13,
  HTTP_REQ_ERR_ERRNO:=14,
  HTTP_REQ_ERR_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_INVALID:=34,
  HTTP_REQ_ERR_CREATE:=35,
  HTTP_REQ_ERR_CREATE_TYPE:=36,

```

```

HTTP_REQ_ERR_CONN:=37,
HTTP_REQ_ERR_CONN_TIMEOUT:=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_TIMEOUT:=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

```

5.1.2.3 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

```

5.1.2.4 E_IotHttpCompressionMode

```

TYPE E_IotHttpCompressionMode:
(
  NoCompression:=0,
  Deflate:=1,
  Gzip:=2
) UDINT;
END_TYPE

```

5.1.3 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 \[► 113\]](#) 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 \[► 113\]](#) 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 \[► 88\]\(\)](#) oder [ParseDocument \[► 89\]\(\)](#) 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](#).

Syntax

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

Ausgänge

Name	Typ
initStatus	HRESULT

☰ **Methoden**

Name	Beschreibung
AddArrayMember [► 49]	Fügt ein Array-Member zu einem JSON-Objekt hinzu.
AddBase64Member [► 49]	Fügt ein Base64-Member zu einem JSON-Objekt hinzu.
AddBoolMember [► 50]	Fügt ein Bool-Member zu einem JSON-Objekt hinzu.
AddDateTimeMember [► 51]	Fügt ein DateTime-Member zu einem JSON-Objekt hinzu.
AddDcTimeMember [► 51]	Fügt ein DcTime-Member zu einem JSON-Objekt hinzu.
AddDoubleMember [► 52]	Fügt ein Double-Member zu einem JSON-Objekt hinzu.
AddFileTimeMember [► 53]	Fügt ein FileTime-Member zu einem JSON-Objekt hinzu.
AddHexBinaryMember [► 53]	Fügt ein HexBinary-Member zu einem JSON-Objekt hinzu.
AddInt64Member [► 54]	Fügt ein Int64-Member zu einem JSON-Objekt hinzu.
AddIntMember [► 55]	Fügt ein Int-Member zu einem JSON-Objekt hinzu.
AddJsonMember [► 56]	Fügt ein JSON-Member zu einem JSON-Objekt hinzu.
AddNullMember [► 56]	Fügt ein NULL-Member zu einem JSON-Objekt hinzu.
AddObjectMember [► 57]	Fügt ein Object-Member zu einem JSON-Objekt hinzu.
AddStringMember [► 57]	Fügt ein String-Member zu einem JSON-Objekt hinzu.
AddUInt64Member [► 58]	Fügt ein UInt64-Member zu einem JSON-Objekt hinzu.
AddUIntMember [► 59]	Fügt ein UInt-Member zu einem JSON-Objekt hinzu.
ArrayBegin [► 60]	Liefert das erste Element eines Arrays.
ArrayEnd [► 60]	Liefert das letzte Element eines Arrays.
ClearArray [► 61]	Löscht den Inhalt eines Arrays.
CopyDocument [► 61]	Kopiert den Inhalt des DOM-Speichers in eine Variable vom Datentyp STRING.
CopyJson [► 63]	Extrahiert ein JSON-Objekt aus einem Key und speichert dieses in einer Variablen vom Datentyp STRING.
CopyString [► 64]	Kopiert den Wert eines Keys in eine Variable vom Datentyp STRING.
FindMember [► 65]	Sucht in einem JSON-Dokument nach einem bestimmten Property.
FindMemberPath [► 65]	Sucht in einem JSON-Dokument nach einem bestimmten Property (Pfad-spezifisch).
GetArraySize [► 67]	Liefert die Anzahl der Elemente in einem JSON-Array.
GetArrayValue [► 67]	Liefert den Wert an der aktuellen Iterator-Position eines Arrays.
GetArrayValueByIdx [► 68]	Liefert den Wert eines Arrays an einem angegebenen Index.
GetBase64 [► 69]	Dekodiert einen Base64-Wert aus einem JSON-Property.
GetBool [► 69]	Liefert den Value eines Properties vom Datentyp BOOL.
GetDateTime [► 70]	Liefert den Value eines Properties vom Datentyp DATE_AND_TIME.
GetDcTime [► 70]	Liefert den Values eines Properties vom Datentyp DCTIME.
GetDocument [► 71]	Gibt den Inhalt des DOM-Speichers als Datentyp STRING(255) zurück.
GetDocumentLength [► 71]	Gibt die Länge eines JSON-Dokuments im DOM-Speicher zurück.
GetDocumentRoot [► 71]	Liefert den Root-Knoten eines JSON-Dokuments im DOM-Speicher.
GetDouble [► 72]	Liefert den Value eines Properties vom Datentyp LREAL.
GetFileTime [► 72]	Liefert den Value eines Properties vom Datentyp DCTIME.
GetHexBinary [► 73]	Dekodiert den HexBinary-Inhalt eines Properties und schreibt diesen an eine bestimmte Speicheradresse,
GetInt [► 73]	Liefert den Value eines Properties vom Datentyp DINT.
GetInt64 [► 74]	Liefert den Value eines Properties vom Datentyp LINT.
GetJson [► 74]	Liefert den Value eines Properties als Datentyp STRING(255) zurück.

Name	Beschreibung
GetJsonLength [► 75]	Liefert die Länge eines Properties, wenn dieses ein JSON-Dokument ist.
GetMaxDecimalPlaces [► 75]	Liefert die aktuelle Einstellung für MaxDecimalPlaces.
GetMemberName [► 75]	Liefert den Namen eines JSON-Property-Members an der Position des aktuellen Iterators,
GetMemberValue [► 76]	Liefert den Value eines JSON-Property-Members an der Position des aktuellen Iterators,
GetString [► 77]	Liefert den Value eines Properties vom Datentyp STRING(255).
GetStringLength [► 77]	Liefert die Länge eines Properties, wenn dessen Value ein String ist.
GetType [► 78]	Liefert den Typ eines Property-Values.
GetUint [► 78]	Liefert den Value eines Properties vom Datentyp UDINT.
GetUint64 [► 79]	Liefert den Value eines Properties vom Datentyp ULINT.
HasMember [► 79]	Prüft, ob ein bestimmtes Property im DOM-Speicher vorhanden ist.
IsArray [► 80]	Prüft, ob es sich bei einem gegebenen Property um ein Array handelt.
IsBase64 [► 80]	Prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp Base64 handelt.
IsBool [► 81]	Prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp BOOL handelt.
IsDouble [► 81]	Prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp Double (SPS: LREAL) handelt.
IsFalse [► 82]	Prüft, ob der Value eines gegebenen Properties FALSE ist.
IsHexBinary [► 82]	Prüft, ob der Value eines Properties ein HexBinary-Format hat.
IsInt [► 83]	Prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp Integer (SPS: DINT) handelt.
IsInt64 [► 83]	Prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp LINT handelt.
IsISO8601TimeFormat [► 83]	Prüft, ob es sich bei dem Value eines gegebenen Properties um ein Zeitformat laut ISO8601 handelt.
IsNull [► 84]	Prüft, ob es sich bei dem Value eines gegebenen Properties um NULL handelt.
IsNumber [► 84]	Prüft, ob es sich bei dem Value eines gegebenen Properties um einen numerischen Wert handelt.
IsObject [► 85]	Prüft, ob es sich bei dem gegebenen Property um ein weiteres JSON-Objekt handelt.
IsString [► 85]	Prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp STRING handelt.
IsTrue [► 86]	Prüft, ob der Wert eines gegebenen Properties TRUE ist.
IsUint [► 86]	Prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp UDINT handelt.
IsUint64 [► 87]	Prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp ULINT handelt.
LoadDocumentFromFile [► 48]	Lädt ein JSON-Dokument aus einer Datei.
MemberBegin [► 87]	Liefert das erste Kindelement unterhalb eines JSON-Properties.
MemberEnd [► 88]	Liefert das letzte Kindelement unterhalb eines JSON-Properties.
NewDocument [► 88]	Erzeugt ein neues, leeres JSON-Dokument im DOM-Speicher.
NextArray [► 88]	Liefert das nächste Element in einem Array.
NextMember [► 89]	Liefert das nächste Property in einem JSON-Dokument.
ParseDocument [► 89]	Lädt ein JSON-Objekt zur weiteren Verarbeitung in den DOM-Speicher.
PopbackValue	Löscht das Element am Ende eines Arrays.

Name	Beschreibung
PushbackBase64Value [► 90]	Hängt einen Base64-Wert an das Ende eines Arrays an.
PushbackBoolValue [► 90]	Hängt einen Base64-Wert an das Ende eines Arrays an.
PushbackDateTimeValue [► 92]	Hängt einen Wert vom Datentyp DATE_AND_TIME an das Ende eines Arrays an.
PushbackDcTimeValue [► 92]	Hängt einen Wert vom Datentyp DCTIME an das Ende eines Arrays an.
PushbackDoubleValue [► 93]	Hängt einen Wert vom Datentyp Double an das Ende eines Arrays an.
PushbackFileTimeValue [► 93]	Hängt einen Wert vom Datentyp FILETIME an das Ende eines Arrays an.
PushbackHexBinaryValue [► 94]	Hängt einen HexBinary-kodierten Wert an das Ende eines Arrays an.
PushbackInt64Value [► 94]	Hängt einen Wert vom Datentyp Int64 an das Ende eines Arrays an.
PushbackIntValue [► 95]	Hängt einen Wert vom Datentyp INT an das Ende eines Arrays an.
PushbackJsonValue [► 96]	Fügt ein JSON-Dokument zum Ende eines Arrays hinzu.
PushbackNullValue [► 96]	Hängt einen NULL-Wert an das Ende eines Arrays an.
PushbackStringValue [► 97]	Hängt einen Wert vom Datentyp String an das Ende eines Arrays an.
PushbackUInt64Value [► 97]	Hängt einen Wert vom Datentyp UInt64 an das Ende eines Arrays an.
PushbackUIntValue [► 98]	Hängt einen Wert vom Datentyp UInt an das Ende eines Arrays an.
RemoveAllMembers [► 98]	Entfernt alle Kindelemente von einem gegebenen Property.
RemoveArray [► 99]	Löscht den Wert des aktuellen Array-Iterators.
RemoveMember [► 100]	Löscht das Property an dem aktuellen Iterator.
RemoveMemberByName [► 101]	Entfernt ein Kindelement von einem gegebenen Property.
SaveDocumentToFile [► 101]	Speichert ein JSON-Dokument in einer Datei.
SetArray [► 103]	Setzt den Value eines Properties auf den Typ „Array“.
SetBase64 [► 103]	Setzt den Value eines Properties auf einen Base64-kodierten Wert.
SetBool [► 104]	Setzt den Value eines Properties auf einen Wert vom Datentyp BOOL.
SetDateTime [► 104]	Setzt den Value eines Properties auf einen Wert vom Datentyp DATE_AND_TIME.
SetDcTime [► 105]	Setzt den Value eines Properties auf einen Wert vom Datentyp DCTIME.
SetDouble [► 105]	Setzt den Value eines Properties auf einen Wert vom Datentyp Double.
SetFileTime [► 106]	Setzt den Value eines Properties auf einen Wert vom Datentyp FILETIME.
SetHexBinary [► 107]	Setzt den Value eines Properties auf einen HexBinary-kodierten Wert.
SetInt [► 107]	Setzt den Value eines Properties auf einen Wert vom Datentyp INT
SetInt64 [► 108]	Setzt den Value eines Properties auf einen Wert vom Datentyp Int64.
SetJson [► 108]	Fügt in den Value eines Properties ein weiteres JSON-Dokument ein.
SetMaxDecimalPlaces [► 109]	Setzt die aktuelle Einstellung für MaxDecimalPlaces.
SetNull [► 109]	Setzt den Value eines Properties auf den Wert NULL.
SetObject [► 110]	Setzt den Value eines Properties auf den Typ „Object“.
SetString [► 110]	Setzt den Value eines Properties auf einen Wert vom Datentyp STRING.
SetUInt [► 111]	Setzt den Value eines Properties auf einen Wert vom Datentyp UInt.
SetUInt64 [► 112]	Setzt den Value eines Properties auf einen Wert vom Datentyp UInt64.

Voraussetzungen

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

5.2.1.1.1 LoadDocumentFromFile

LoadDocumentFromFile	
sFile <i>STRING</i>	<i>BOOL</i> LoadDocumentFromFile
bExec <i>REFERENCE TO BOOL</i>	<i>HRESULT</i> hrErrorCode

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.

● Encoding

i Bitte beachten Sie, dass die eingelesenen Dateien in UTF-8 formatiert sein müssen. Formate wie bspw. UTF-16 werden nicht unterstützt.

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
```

👉 Rückgabewert

Name	Typ
LoadDocumentFromFile	BOOL

👉 Eingänge

Name	Typ
bExec	REFERENCE TO BOOL

👉 / 👉 Ein-/Ausgänge

Name	Typ
sFile	STRING

👉 Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

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

5.2.1.1.2 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
```

Rückgabewert

Name	Typ
AddArrayMember	SJsonValue (mögliche Ungültigkeit beachten)

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
reserve	UDINT

Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

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

5.2.1.1.3 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
```

Rückgabewert

Name	Typ
AddBase64Member	SJsonValue (mögliche Ungültigkeit beachten)

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
p	PVOID
n	DINT

/ Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

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

5.2.1.1.4 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
```

Rückgabewert

Name	Typ
AddBoolMember	SJsonValue (mögliche Ungültigkeit beachten)

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
value	BOOL

 /  Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

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

5.2.1.1.5 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
```

 Rückgabewert

Name	Typ
AddDateTimeMember	SJsonValue (mögliche Ungültigkeit beachten)

 Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
value	DATE_AND_TIME

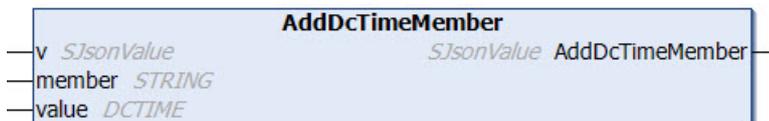
 /  Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

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

5.2.1.1.6 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

```

🔑 Rückgabewert

Name	Typ
AddDcTimeMember	SJsonValue (mögliche Ungültigkeit beachten)

🔑 Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
value	DCTIME

🔑 / 🔑 Ein-/Ausgänge

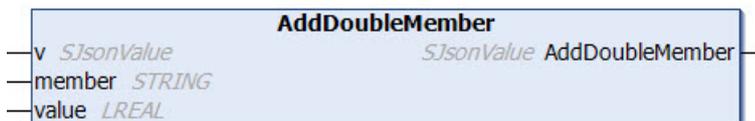
Name	Typ
member	STRING

Beispielaufruf:

```

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

```

5.2.1.1.7 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

```

🔑 Rückgabewert

Name	Typ
AddDoubleMember	SJsonValue (mögliche Ungültigkeit beachten)

🔑 Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
value	LREAL

 /  Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

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

5.2.1.1.8 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
```

 Rückgabewert

Name	Typ
AddFileTimeMember	SJsonValue (mögliche Ungültigkeit beachten)

 Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
value	FILETIME

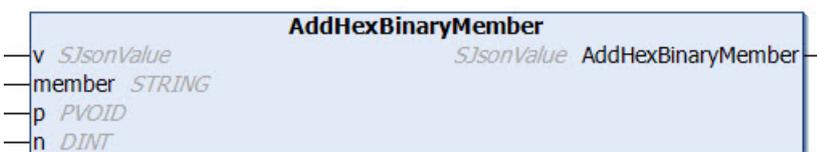
 /  Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

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

5.2.1.1.9 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

```

📌 Rückgabewert

Name	Typ
AddHexBinaryMember	SJsonValue (mögliche Ungültigkeit beachten)

📌 Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
p	PVOID
n	DINT

📌 / 📌 Ein-/Ausgänge

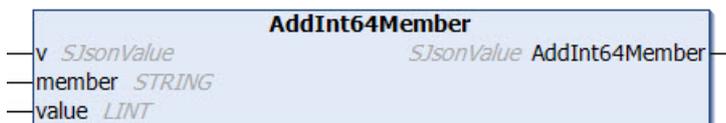
Name	Typ
member	STRING

Beispielaufruf:

```

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

```

5.2.1.1.10 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

```

📌 Rückgabewert

Name	Typ
AddInt64Member	SJsonValue (mögliche Ungültigkeit beachten)

 **Eingänge**

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
value	LINT

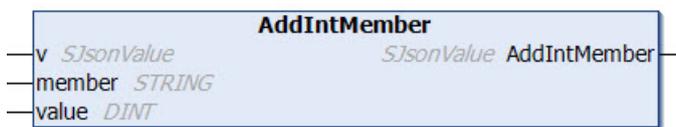
 **Ein-/Ausgänge**

Name	Typ
member	STRING

Beispielaufruf:

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

5.2.1.1.11 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
```

 **Rückgabewert**

Name	Typ
AddIntMember	SJsonValue (mögliche Ungültigkeit beachten)

 **Eingänge**

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
value	DINT

 **Ein-/Ausgänge**

Name	Typ
member	STRING

Beispielaufruf:

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

5.2.1.1.12 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
```

Rückgabewert

Name	Typ
AddJsonMember	SJsonValue (mögliche Ungültigkeit beachten)

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

Ein-/Ausgänge

Name	Typ
member	STRING
rawJson	STRING

Beispielaufruf:

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

5.2.1.1.13 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
```

Rückgabewert

Name	Typ
AddNullMember	SJsonValue (mögliche Ungültigkeit beachten)

 **Eingänge**

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

 /  **Ein-/Ausgänge**

Name	Typ
member	STRING

Beispielaufruf:

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

5.2.1.1.14 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
```

 **Rückgabewert**

Name	Typ
AddObjectMember	SJsonValue (mögliche Ungültigkeit beachten)

 **Eingänge**

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

 /  **Ein-/Ausgänge**

Name	Typ
member	STRING

Beispielaufruf:

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

5.2.1.1.15 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
```

👉 Rückgabewert

Name	Typ
AddStringMember	SJsonValue (mögliche Ungültigkeit beachten)

👉 Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

👉 / 👈 Ein-/Ausgänge

Name	Typ
member	STRING
value	STRING

Beispielaufruf:

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

5.2.1.1.16 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
```

👉 Rückgabewert

Name	Typ
AddUInt64Member	SJsonValue (mögliche Ungültigkeit beachten)

 **Eingänge**

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
value	ULINT

 **Ein-/Ausgänge**

Name	Typ
member	STRING

Beispielaufruf:

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

5.2.1.1.17 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
```

 **Rückgabewert**

Name	Typ
AddUintMember	SJsonValue (mögliche Ungültigkeit beachten)

 **Eingänge**

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
value	UDINT

 **Ein-/Ausgänge**

Name	Typ
member	STRING

Beispielaufruf:

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

5.2.1.1.18 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 : SJsonValueIterator
VAR_INPUT
  v : SJsonValue;
END_VAR
  
```

Rückgabewert

Name	Typ
ArrayBegin	SJsonValueIterator

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

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 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 : SJsonValueIterator
VAR_INPUT
  v : SJsonValue;
END_VAR
  
```

Rückgabewert

Name	Typ
ArrayEnd	SJsonValueIterator

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

Beispielaufruf:

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

5.2.1.1.20 ClearArray



Diese Methode löscht den Inhalt eines Arrays.

Syntax

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

Rückgabewert

Name	Typ
ClearArray	BOOL

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
i	SJsonAlterator

Beispielaufruf:

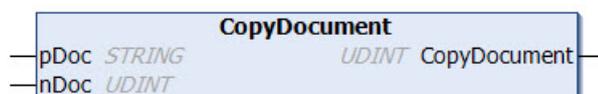
Gegeben sei das folgende JSON-Dokument, das in den DOM-Speicher geladen wird:

```
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
```

5.2.1.1.21 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
```

Rückgabewert

Name	Typ
CopyDocument	UDINT

Eingänge

Name	Typ
nDoc	DINT

/ Ein-/Ausgänge

Name	Typ
pDoc	STRING

Beispielaufruf:

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

5.2.1.1.22 CopyFrom



Syntax

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

Rückgabewert

Name	Typ
CopyFrom	SJsonValue (mögliche Ungültigkeit beachten)

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
w	SJsonValue (mögliche Ungültigkeit beachten)

5.2.1.1.23 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
    
```

Rückgabewert

Name	Typ
CopyJson	UDINT

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

Ein-/Ausgänge

Name	Typ
pDoc	STRING
nDoc	STRING

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"}
```

5.2.1.1.24 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
```

Rückgabewert

Name	Typ
CopyString	UDINT

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

Ein-/Ausgänge

Name	Typ
pStr	STRING
nStr	UDINT

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
```

5.2.1.1.25 ExceptionRaised



Syntax

```
METHOD ExceptionRaised : BOOL
```

Rückgabewert

Name	Typ
ExceptionRaised	BOOL

5.2.1.1.26 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
```

Rückgabewert

Name	Typ
FindMember	SJsonValue (mögliche Ungültigkeit beachten)

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

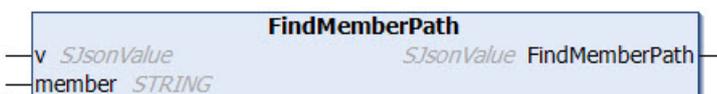
Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

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

5.2.1.1.27 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
```

Rückgabewert

Name	Typ
FindMemberPath	SJsonValue (mögliche Ungültigkeit beachten)

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

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

Der Zugriff auf verschachtelte Objekte funktioniert nach dem Schema *a/b/c*, um eine Variable in einer JSON-Hierarchie zu finden. Der Aufruf für die Variable *c* des folgenden JSON-Dokuments lautet:

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

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

Unterstützung für Arrays

Die Methode unterstützt JSON-Dokumente mit Arrays ab TwinCAT-Version >3.1.4024.35. Mithilfe des Zeichens # kann auf Elemente eines Arrays zugegriffen werden.

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

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

```
  },  
]
```

Der Beispielaufruf greift auf das zweite Element des äußeren Arrays zu (#1), anschließend auf das dritte Element des Arrays unter dem Unterelement *Third*.

Behandlung von Sonderfällen

Das Einfügen des Zeichens ~ sorgt für die Sonderbehandlung innerhalb eines Pfades. Die folgende Tabelle listet die verschiedenen Möglichkeiten auf.

Ausdruck	Ergebnis	Beispiel
~0	~ wird als Zeichen im String verwendet.	Test/Hallo~0123 wird zu Test/Hallo~123
~1	Der Ausdruck wird durch das Zeichen / ersetzt, was nicht als Trenner, sondern als Teil des Strings interpretiert wird.	Test/Hallo~1123 wird Test/Hallo/123
~2	Der Ausdruck wird durch das Zeichen # ersetzt, was nicht als Array-Index, sondern als Teil des Strings interpretiert wird.	Test/Hallo~2123 wird zu Test/Hallo#123

5.2.1.1.28 GetArraySize



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

Syntax

```

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

Rückgabewert

Name	Typ
GetArraySize	UDINT

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

Beispielaufruf:

```

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

5.2.1.1.29 GetArrayValue



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

Syntax

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

📌 Rückgabewert

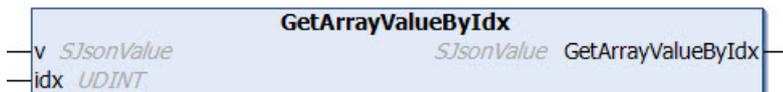
Name	Typ
GetArrayValue	SJsonValue (mögliche Ungültigkeit beachten)

📌 Eingänge

Name	Typ
i	SJsonAIterator

Beispielaufruf:

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

5.2.1.1.30 GetArrayValueByIdx

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

Syntax

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

📌 Rückgabewert

Name	Typ
GetArrayValueByIdx	SJsonValue (mögliche Ungültigkeit beachten)

📌 Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
idx	UDINT

Beispielaufruf:

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

5.2.1.1.31 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
```

Rückgabewert

Name	Typ
GetBase64	DINT

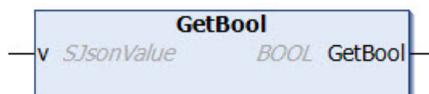
Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
p	PVOID
n	DINT

Beispielaufruf:

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

5.2.1.1.32 GetBool



Diese Methode liefert den Value eines Properties vom Datentyp BOOL.

Syntax

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

Rückgabewert

Name	Typ
GetBool	BOOL

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

5.2.1.1.33 **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
```

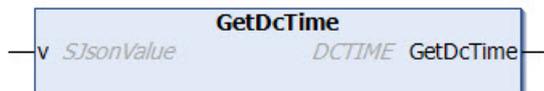
Rückgabewert

Name	Typ
GetDateTime	DATE_AND_TIME

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

5.2.1.1.34 **GetDcTime**



Diese Methode liefert den Values eines Properties vom Datentyp DCTIME.

Syntax

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

Rückgabewert

Name	Typ
GetDcTime	DCTIME

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

Beispielaufruf:

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

5.2.1.1.35 GetDocument

GetDocument

STRING(255) 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 [▶ 61]()` verwendet werden.

Syntax

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

Rückgabewert

Name	Typ
GetDocument	STRING(255)

Beispielaufruf:

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

5.2.1.1.36 GetDocumentLength

GetDocumentLength

UDINT GetDocumentLength

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

Syntax

```
METHOD GetDocumentLength: UDINT
```

Rückgabewert

Name	Typ
GetDocumentLength	UDINT

Beispielaufruf:

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

5.2.1.1.37 GetDocumentRoot

GetDocumentRoot

SJsonValue GetDocumentRoot

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

Syntax

```
METHOD GetDocumentRoot : SJsonValue
```

Rückgabewert

Name	Typ
GetDocumentRoot	SJsonValue (mögliche Ungültigkeit beachten)

Beispielaufruf:

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

5.2.1.1.38 GetDouble



Diese Methode liefert den Value eines Properties vom Datentyp LREAL.

Syntax

```

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

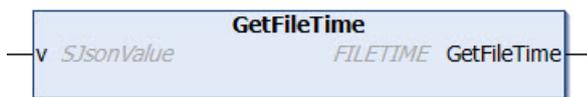
Rückgabewert

Name	Typ
GetDouble	LREAL

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

5.2.1.1.39 GetFileTime



Diese Methode liefert den Value eines Properties vom Datentyp DCTIME.

Syntax

```

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

Rückgabewert

Name	Typ
GetFileTime	FILETIME

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

Beispielaufruf:

```

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

5.2.1.1.40 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
```

Rückgabewert

Name	Typ
GetHexBinary	DINT

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
p	PVOID
n	DINT

Beispielaufruf:

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

5.2.1.1.41 GetInt



Diese Methode liefert den Value eines Properties vom Datentyp DINT.

Syntax

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

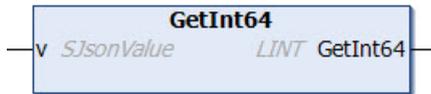
Rückgabewert

Name	Typ
GetInt	DINT

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

5.2.1.1.42 GetInt64



Diese Methode liefert den Value eines Properties vom Datentyp LINT.

Syntax

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

Rückgabewert

Name	Typ
GetInt64	LINT

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

5.2.1.1.43 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](#) [\[▶ 63\]](#)() verwendet werden.

Syntax

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

Rückgabewert

Name	Typ
GetJson	STRING(255)

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

Beispielaufruf:

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

5.2.1.1.44 GetJsonLength



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

Syntax

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

Rückgabewert

Name	Typ
GetJsonLength	UDINT

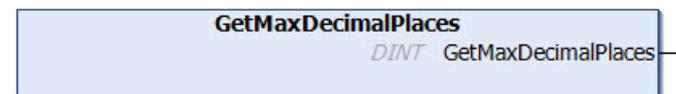
Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

Beispielaufruf:

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

5.2.1.1.45 GetMaxDecimalPlaces



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

Syntax

```
METHOD GetMaxDecimalPlaces : DINT
```

Rückgabewert

Name	Typ
GetMaxDecimalPlaces	DINT

Beispielaufruf:

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

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

📌 Rückgabewert

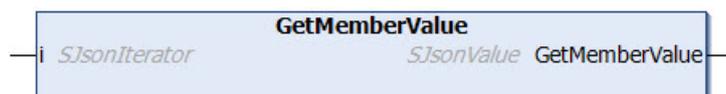
Name	Typ
GetMemberName	STRING(255)

📌 Eingänge

Name	Typ
i	SJsonIterator

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
```

5.2.1.1.47 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
```

📌 Rückgabewert

Name	Typ
GetMemberValue	SJsonValue (mögliche Ungültigkeit beachten)

📌 Eingänge

Name	Typ
i	SJsonIterator

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
```

5.2.1.1.48 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](#) (▶ 64)() verwendet werden.

Syntax

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

Rückgabewert

Name	Typ
GetString	STRING(255)

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

5.2.1.1.49 GetStringLength



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

Syntax

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

Rückgabewert

Name	Typ
GetStringLength	UDINT

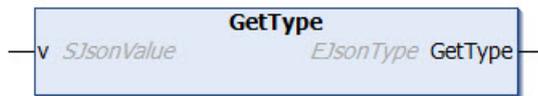
Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

Beispielaufruf:

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

5.2.1.1.50 GetType



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

Syntax

```

METHOD GetType : EJsonType
VAR_INPUT
  v          : SJsonValue
END_VAR

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

Rückgabewert

Name	Typ
GetType	EJsonType

Eingänge

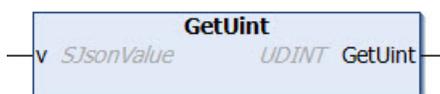
Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

Beispielaufruf:

```

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

5.2.1.1.51 GetUint



Diese Methode liefert den Value eines Properties vom Datentyp UDINT.

Syntax

```

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

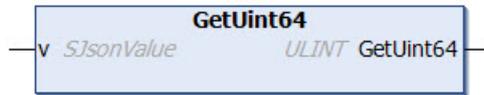
Rückgabewert

Name	Typ
GetUint	UDINT

 **Eingänge**

Name	Typ
v	SjsonValue (mögliche Ungültigkeit beachten)

5.2.1.1.52 GetUint64



Diese Methode liefert den Value eines Properties vom Datentyp ULINT.

Syntax

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

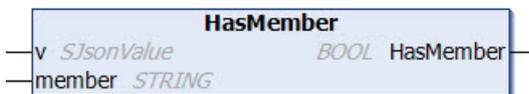
 **Rückgabewert**

Name	Typ
GetUint64	ULINT

 **Eingänge**

Name	Typ
v	SjsonValue (mögliche Ungültigkeit beachten)

5.2.1.1.53 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
```

 **Rückgabewert**

Name	Typ
HasMember	BOOL

 **Eingänge**

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

📁 / 📁 Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

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

5.2.1.1.54 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
```

📁 Rückgabewert

Name	Typ
IsArray	BOOL

📁 Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

5.2.1.1.55 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
```

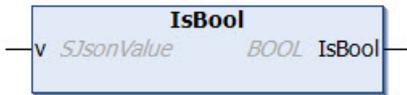
📁 Rückgabewert

Name	Typ
IsBase64	BOOL

 Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

5.2.1.1.56 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
```

 Rückgabewert

Name	Typ
IsBool	BOOL

 Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

5.2.1.1.57 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
```

 Rückgabewert

Name	Typ
IsDouble	BOOL

 Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

5.2.1.1.58 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
  
```

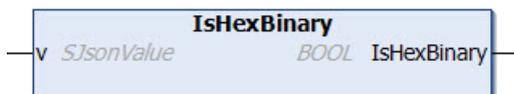
Rückgabewert

Name	Typ
IsFalse	BOOL

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

5.2.1.1.59 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
  
```

Rückgabewert

Name	Typ
IsHexBinary	BOOL

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

Beispielaufruf:

```

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

5.2.1.1.60 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
```

Rückgabewert

Name	Typ
IsInt	BOOL

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

5.2.1.1.61 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
```

Rückgabewert

Name	Typ
IsInt64	BOOL

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

5.2.1.1.62 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
```

📌 Rückgabewert

Name	Typ
IsISO8601TimeFormat	BOOL

📌 Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

5.2.1.1.63 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

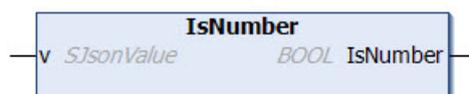
```
METHOD IsNull : BOOL
VAR_INPUT
  v : SJsonValue;
END_VAR
```

📌 Rückgabewert

Name	Typ
IsNull	BOOL

📌 Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

5.2.1.1.64 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
```

 Rückgabewert

Name	Typ
IsNumber	BOOL

 Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

5.2.1.1.65 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
```

 Rückgabewert

Name	Typ
IsObject	BOOL

 Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

5.2.1.1.66 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
```

 Rückgabewert

Name	Typ
IsString	BOOL

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

5.2.1.1.67 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
```

Rückgabewert

Name	Typ
IsTrue	BOOL

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

5.2.1.1.68 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
```

Rückgabewert

Name	Typ
IsUInt	BOOL

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

5.2.1.1.69 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
```

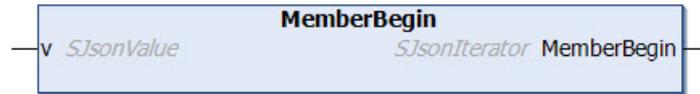
Rückgabewert

Name	Typ
IsUint64	BOOL

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

5.2.1.1.70 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
```

Rückgabewert

Name	Typ
MemberBegin	SJsonIterator

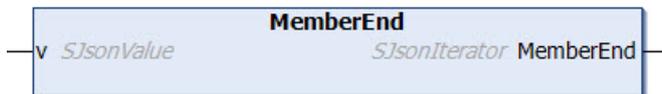
Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

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
```

5.2.1.1.71 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
```

Rückgabewert

Name	Typ
MemberEnd	SJsonIterator

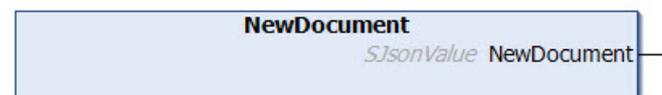
Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

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
```

5.2.1.1.72 NewDocument



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

Syntax

```
METHOD NewDocument : SJsonValue
```

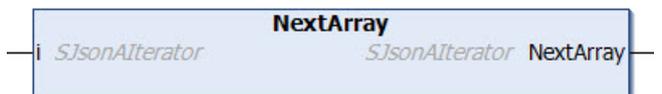
Beispielaufruf:

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

Rückgabewert

Name	Typ
NewDocument	SJsonValue (mögliche Ungültigkeit beachten)

5.2.1.1.73 NextArray



Syntax

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

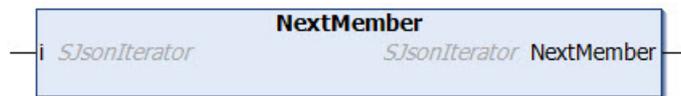
 **Rückgabewert**

Name	Typ
NextArray	SJsonIterator

 **Eingänge**

Name	Typ
i	SJsonIterator

5.2.1.1.74 NextMember



Diese Methode liefert das nächste Kindelement unterhalb eines JSON-Properties und kann zusammen mit den Methoden [MemberBegin \[▶ 87\]\(\)](#) und [MemberEnd \[▶ 88\]\(\)](#) zur Iteration durch ein JSON-Property verwendet werden.

Syntax

```
METHOD NextMember : SJsonIterator
VAR_INPUT
  v : SJsonIterator;
END_VAR
```

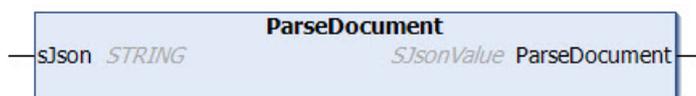
 **Rückgabewert**

Name	Typ
NextMember	SJsonIterator

 **Eingänge**

Name	Typ
i	SJsonIterator

5.2.1.1.75 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
```

📌 Rückgabewert

Name	Typ
ParseDocument	SJsonValue (mögliche Ungültigkeit beachten)

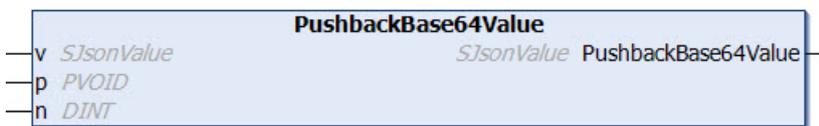
📌 Eingänge

Name	Typ
sJson	STRING

Beispielaufruf:

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

5.2.1.1.76 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
```

📌 Rückgabewert

Name	Typ
PushbackBase64Value	SJsonValue (mögliche Ungültigkeit beachten)

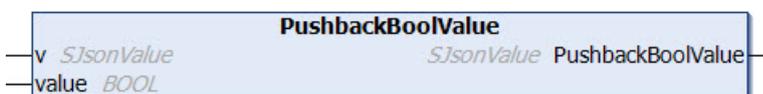
📌 Eingänge

Name	Typ
v	sJsonValue (mögliche Ungültigkeit beachten)
p	PVOID
n	DINT

Beispielaufruf:

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

5.2.1.1.77 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
```

 **Rückgabewert**

Name	Typ
PushbackBoolValue	SJsonValue (mögliche Ungültigkeit beachten)

 **Eingänge**

Name	Typ
v	sJsonValue (mögliche Ungültigkeit beachten)
value	BOOL

Beispielaufruf:

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

5.2.1.1.78 PushbackCopyValue



Diese Methode fügt ein JSON-Dokument aus dem Speicher eines [FB JsonDomParser](#) [► 42] zum Ende eines Arrays hinzu.

Syntax

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

 **Rückgabewert**

Name	Typ
PushbackCopyValue	SJsonValue (mögliche Ungültigkeit beachten)

 **Eingänge**

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
json	SJsonValue (mögliche Ungültigkeit beachten)

Beispielaufruf:

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

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

5.2.1.1.79 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

```

Rückgabewert

Name	Typ
PushbackDateTimeValue	SJsonValue (mögliche Ungültigkeit beachten)

Eingänge

Name	Typ
v	sJsonValue (mögliche Ungültigkeit beachten)
value	DATE_AND_TIME

Beispielaufruf:

```

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

```

5.2.1.1.80 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

```

Rückgabewert

Name	Typ
PushbackDcTimeValue	SJsonValue (mögliche Ungültigkeit beachten)

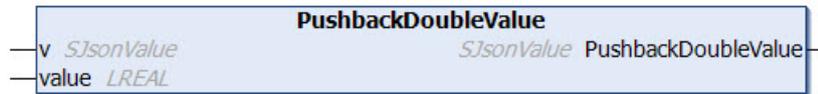
Eingänge

Name	Typ
v	sJsonValue (mögliche Ungültigkeit beachten)
value	DCTIME

Beispielaufruf:

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

5.2.1.1.81 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
```

Rückgabewert

Name	Typ
PushbackDoubleValue	SJsonValue (mögliche Ungültigkeit beachten)

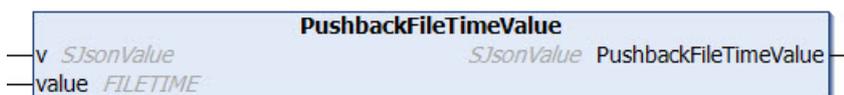
Eingänge

Name	Typ
v	sJsonValue (mögliche Ungültigkeit beachten)
value	LREAL

Beispielaufruf:

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

5.2.1.1.82 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
```

Rückgabewert

Name	Typ
PushbackFileTimeValue	SJsonValue (mögliche Ungültigkeit beachten)

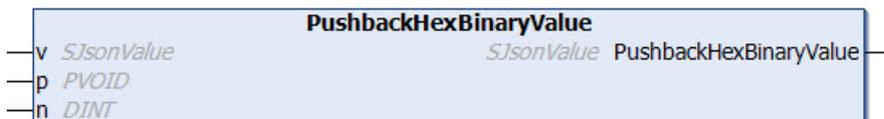
Eingänge

Name	Typ
v	sJsonValue (mögliche Ungültigkeit beachten)
value	FILETIME

Beispielaufruf:

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

5.2.1.1.83 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
```

Rückgabewert

Name	Typ
PushbackHexBinaryValue	SJsonValue (mögliche Ungültigkeit beachten)

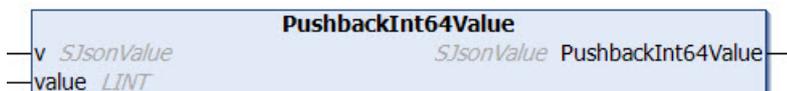
Eingänge

Name	Typ
v	sJsonValue (mögliche Ungültigkeit beachten)
p	PVOID
n	DINT

Beispielaufruf:

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

5.2.1.1.84 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
```

 **Rückgabewert**

Name	Typ
PushbackInt64Value	SJsonValue (mögliche Ungültigkeit beachten)

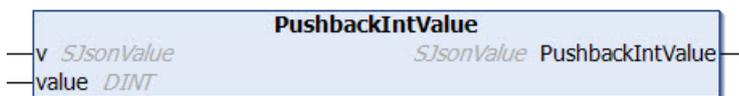
 **Eingänge**

Name	Typ
v	sJsonValue (mögliche Ungültigkeit beachten)
value	LINT

Beispielaufruf:

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

5.2.1.1.85 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
```

 **Rückgabewert**

Name	Typ
PushbackIntValue	SJsonValue (mögliche Ungültigkeit beachten)

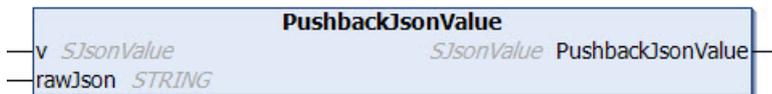
 **Eingänge**

Name	Typ
v	sJsonValue (mögliche Ungültigkeit beachten)
value	DINT

Beispielaufruf:

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

5.2.1.1.86 PushbackJsonValue



Diese Methode fügt ein als STRING in der SPS vorhandenes 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
  
```

Rückgabewert

Name	Typ
PushbackJsonValue	SJsonValue (mögliche Ungültigkeit beachten)

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

Ein-/Ausgänge

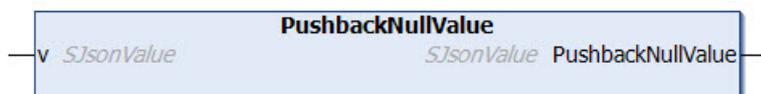
Name	Typ
rawJson	STRING

Beispielaufruf:

```

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

5.2.1.1.87 PushbackNullValue



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

Syntax

```

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

Rückgabewert

Name	Typ
PushbackNullValue	SJsonValue (mögliche Ungültigkeit beachten)

 **Eingänge**

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

Beispielaufruf:

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

5.2.1.1.88 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
```

 **Rückgabewert**

Name	Typ
PushbackStringValue	SJsonValue (mögliche Ungültigkeit beachten)

 **Eingänge**

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

 /  **Ein-/Ausgänge**

Name	Typ
value	STRING

Beispielaufruf:

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

5.2.1.1.89 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

```

📌 Rückgabewert

Name	Typ
PushbackUint64Value	SJsonValue (mögliche Ungültigkeit beachten)

📌 Eingänge

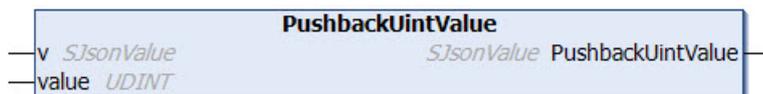
Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
value	ULINT

Beispielaufruf:

```

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

```

5.2.1.1.90 PushbackUintValue

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

Syntax

```

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

```

📌 Rückgabewert

Name	Typ
PushbackUintValue	SJsonValue (mögliche Ungültigkeit beachten)

📌 Eingänge

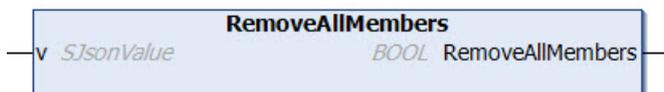
Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
value	UDINT

Beispielaufruf:

```

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

```

5.2.1.1.91 RemoveAllMembers

Diese Methode entfernt alle Kindelemente von einem gegebenen Property.

Syntax

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

 **Rückgabewert**

Name	Typ
RemoveAllMembers	BOOL

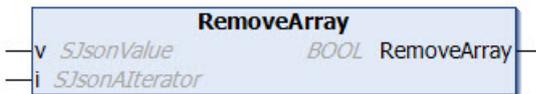
 **Eingänge**

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

Beispielaufruf:

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

5.2.1.1.92 RemoveArray



Diese Methode löscht den Wert des aktuellen Array-Iterators.

Syntax

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

 **Rückgabewert**

Name	Typ
RemoveArray	BOOL

 **Eingänge**

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
i	SJsonAIterator

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

```

5.2.1.1.93 RemoveMember



Diese Methode löscht das Property an dem aktuellen Iterator.

Syntax

```

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

```

Rückgabewert

Name	Typ
RemoveMember	BOOL

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
i	SJsonIterator
keepOrder	BOOL

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]}';
```

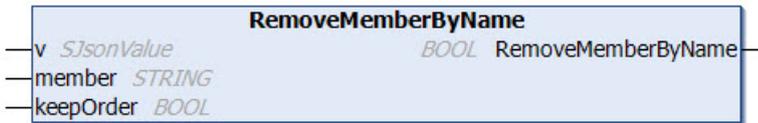
Das Property „array“ soll gelöscht werden. Zunächst wird das JSON-Dokument iterativ nach dem Property „array“ durchsucht, anschließend wird es entfernt.

```

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

```

5.2.1.1.94 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
```

Rückgabewert

Name	Typ
RemoveMemberByName	BOOL

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
keepOrder	BOOL

Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

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

5.2.1.1.95 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
```

Rückgabewert

Name	Typ
SaveDocumentToFile	BOOL

Eingänge

Name	Typ
bExec	REFERENCE TO BOOL

/ Ein-/Ausgänge

Name	Typ
sFile	STRING

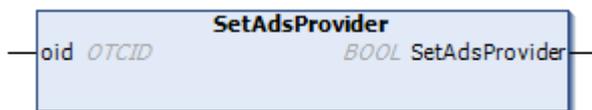
Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

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

5.2.1.1.96 SetAdsProvider



Diese Methode setzt den Kontext, in dem die File Access-Zugriffe beim Laden und Speichern der JSON-Dokumente abgearbeitet werden. Wenn kein ADS-Provider angegeben wird, wird die aktuelle Task genutzt.

Syntax

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

Rückgabewert

Name	Typ	Beschreibung
SetAdsProvider	BOOL	Die Methode gibt TRUE zurück, wenn das Setzen des ADS-Providers erfolgreich war.

Eingänge

Name	Typ	Beschreibung
oid	OTCID	Object ID der Task.

Beispielaufruf:

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

5.2.1.1.97 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
```

Rückgabewert

Name	Typ
SetArray	SJsonValue (mögliche Ungültigkeit beachten)

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

Beispielaufruf:

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

5.2.1.1.98 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
```

Rückgabewert

Name	Typ
SetBase64	SJsonValue (mögliche Ungültigkeit beachten)

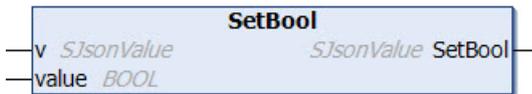
🚩 Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
p	PVOID
n	DINT

Beispielaufruf:

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

5.2.1.1.99 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
```

🚩 Rückgabewert

Name	Typ
SetBool	SJsonValue (mögliche Ungültigkeit beachten)

🚩 Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
value	BOOL

Beispielaufruf:

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

5.2.1.1.100 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
```

 Rückgabewert

Name	Typ
SetDateTime	SJsonValue (mögliche Ungültigkeit beachten)

 Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
value	DATE_AND_TIME

Beispielaufruf:

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

5.2.1.1.101 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
```

 Rückgabewert

Name	Typ
SetDcTime	SJsonValue (mögliche Ungültigkeit beachten)

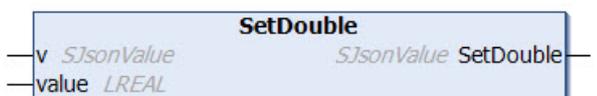
 Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
value	DCTIME

Beispielaufruf:

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

5.2.1.1.102 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

```

📌 Rückgabewert

Name	Typ
SetDouble	SJsonValue (mögliche Ungültigkeit beachten)

📌 Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
value	LREAL

Beispielaufruf:

```

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

```

5.2.1.1.103 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

```

📌 Rückgabewert

Name	Typ
SetFileTime	SJsonValue (mögliche Ungültigkeit beachten)

📌 Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
value	FILETIME

Beispielaufruf:

```

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

```

5.2.1.1.104 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
```

Rückgabewert

Name	Typ
SetHexBinary	SJsonValue (mögliche Ungültigkeit beachten)

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
p	PVOID
n	DINT

Beispielaufruf:

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

5.2.1.1.105 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
```

Rückgabewert

Name	Typ
SetInt	SJsonValue (mögliche Ungültigkeit beachten)

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
value	DINT

Beispielaufruf:

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

5.2.1.1.106 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
```

Rückgabewert

Name	Typ
SetInt64	SJsonValue (mögliche Ungültigkeit beachten)

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
value	LINT

Beispielaufruf:

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

5.2.1.1.107 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
```

 Rückgabewert

Name	Typ
SetJson	SJsonValue (mögliche Ungültigkeit beachten)

 Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

 /  Ein-/Ausgänge

Name	Typ
rawJson	STRING

Beispielaufruf:

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

5.2.1.1.108 SetMaxDecimalPlaces

Diese Funktion legt die Anzahl an Nachkommastellen fest, nach denen eine Gleitkommazahl gerundet wird.



Syntax

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

 Eingänge

Name	Typ
dp	DINT

Beispielaufruf:

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

5.2.1.1.109 SetNull



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

Syntax

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

📌 Rückgabewert

Name	Typ
SetNull	SJsonValue (mögliche Ungültigkeit beachten)

📌 Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

Beispielaufruf:

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

5.2.1.1.110 SetObject



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

Syntax

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

📌 Rückgabewert

Name	Typ
SetObject	SJsonValue (mögliche Ungültigkeit beachten)

📌 Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

Beispielaufruf:

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

5.2.1.1.111 SetString



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

Syntax

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

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

 Rückgabewert

Name	Typ
SetString	SJsonValue (mögliche Ungültigkeit beachten)

 Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)

 Ein-/Ausgänge

Name	Typ
value	STRING

Beispielaufruf:

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

5.2.1.1.112 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
```

 Rückgabewert

Name	Typ
SetUInt	SJsonValue (mögliche Ungültigkeit beachten)

 Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
value	UDINT

Beispielaufruf:

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

5.2.1.1.113 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
```

Rückgabewert

Name	Typ
SetUint64	SJsonValue (mögliche Ungültigkeit beachten)

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
value	ULINT

Beispielaufruf:

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

5.2.1.1.114 Swap



Diese Methode tauscht den Inhalt von zwei Pointern.

Syntax

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

Rückgabewert

Name	Typ
Swap	BOOL

Eingänge

Name	Typ
v	SJsonValue (mögliche Ungültigkeit beachten)
w	SJsonValue (mögliche Ungültigkeit beachten)

5.2.1.2 FB_JsonDynDomParser



Dieser Funktionsblock ist von demselben internen Funktionsbaustein abgeleitet wie der [FB_JsonDomParser](#) [▶ 42] 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](#) [▶ 42] eine bessere Performance ermöglicht.

● Strings im UTF-8-Format

I 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](#).

Syntax

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

👉 Ausgänge

Name	Typ
initStatus	HRESULT

🔗 Methoden

Alle Methoden finden Sie in [FB_JsonDomParser](#) [▶ 42].

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



Syntax

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

Ausgänge

Name	Typ
initStatus	HRESULT

Methoden

Name	Beschreibung
DecodeBase64 [▶ 114]	Konvertiert einen Base64-formatierten String in Binärdaten.
DecodeDateTime [▶ 115]	Erzeugt aus einem genormten ISO8601-Zeitformat (z. B. YYYY-MM-DDThh:mm:ss) eine SPS-Variable vom Datentyp DATE_AND_TIME bzw. DT.
DecodeDcTime [▶ 116]	Erzeugt aus einem genormten ISO8601 Zeitformat (z. B. YYYY-MM-DDThh:mm:ss) eine SPS-Variable vom Datentyp DCTIME.
DecodeFileTime [▶ 116]	Erzeugt aus einem genormten ISO8601-Zeitformat (z. B. YYYY-MM-DDThh:mm:ss) eine SPS-Variable vom Datentyp FILETIME.
DecodeHexBinary [▶ 117]	Konvertiert einen String, der Hexadezimalwerte enthält, in Binärdaten um.
IsBase64 [▶ 118]	Prüft, ob der übergebene String dem Base64-Format entspricht.
IsHexBinary [▶ 118]	Prüft, ob der übergebene String aus Hexadezimalwerten besteht.
IsISO8601TimeFormat [▶ 119]	Prüft, ob der übergebene String dem genormten ISO8601-Zeitformat entspricht.
Parse [▶ 119]	Startet den SAX-Reader-Parsing-Vorgang.
ParseValues [▶ 120]	Startet den SAX-Reader-Parsing-Vorgang.

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
```

Rückgabewert

Name	Typ
DecodeBase64	BOOL

 **Eingänge**

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

 **Ausgänge**

Name	Typ
hrErrorCode	HRESULT

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
```

 **Rückgabewert**

Name	Typ
DecodeDateTime	BOOL

 **Ein-/Ausgänge**

Name	Typ
sDT	STRING

 **Ausgänge**

Name	Typ
nDT	DATE_AND_TIME
hrErrorCode	HRESULT

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
```

Rückgabewert

Name	Typ
DecodeDcTime	BOOL

/ Ein-/Ausgänge

Name	Typ
sDC	STRING

Ausgänge

Name	Typ
nDC	DCTIME
hrErrorCode	HRESULT

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
```

```
nFT      : FILETIME;
hrErrorCode : HRESULT;
END_VAR
```

 Rückgabewert

Name	Typ
DecodeDateTime	BOOL

 /  Ein-/Ausgänge

Name	Typ
sFT	STRING

 Ausgänge

Name	Typ
nFT	FILETIME
hrErrorCode	HRESULT

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
```

 Rückgabewert

Name	Typ
DecodeHexBinary	BOOL

 Eingänge

Name	Typ
pBytes	POINTER TO BYTE
nBytes	REFERENCE TO DINT

📁 / 📁 Ein-/Ausgänge

Name	Typ
sHex	STRING

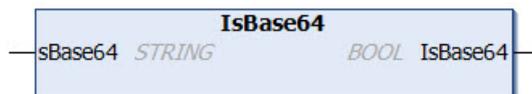
📁 Ausgänge

Name	Typ
hrErrorCode	HRESULT

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
```

📁 Rückgabewert

Name	Typ
IsBase64	BOOL

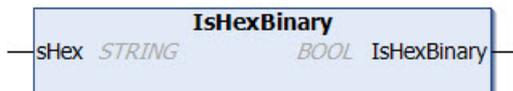
📁 / 📁 Ein-/Ausgänge

Name	Typ
sBase64	STRING

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
```

 Rückgabewert

Name	Typ
IsHexBinary	BOOL

 /  Ein-/Ausgänge

Name	Typ
sHex	STRING

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
```

 Rückgabewert

Name	Typ
IsISO8601TimeFormat	BOOL

 /  Ein-/Ausgänge

Name	Typ
sDT	STRING

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

```

Rückgabewert

Name	Typ
Parse	BOOL

Eingänge

Name	Typ
ipHdl	ITcJsonSaxHandler

Ein-/Ausgänge

Name	Typ
sJson	STRING

Ausgänge

Name	Typ
hrErrorCode	HRESULT

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

```

Rückgabewert

Name	Typ
ParseValues	BOOL

 Eingänge

Name	Typ
ipHdl	ITcJsonSaxValues

 /  Ein-/Ausgänge

Name	Typ
sJson	STRING

 Ausgänge

Name	Typ
hrErrorCode	HRESULT

5.2.1.4 FB_JsonSaxWriter



i 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](#).

Syntax

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

 Ausgänge

Name	Typ
initStatus	HRESULT

☰ **Methoden**

Name	Beschreibung
AddBase64 [▶ 124]	Fügt einen Wert vom Datentyp Base64 zu einem Property hinzu.
AddBool [▶ 124]	Fügt einen Wert vom Datentyp BOOL zu einem Property hinzu.
AddDateTime [▶ 125]	Fügt einen Wert vom Datentyp DATE_AND_TIME zu einem Property hinzu.
AddDcTime [▶ 125]	Fügt einen Wert vom Datentyp DCTIME zu einem Property hinzu.
AddDint [▶ 125]	Fügt einen Wert vom Datentyp DINT zu einem Property hinzu.
AddFileTime [▶ 126]	Fügt einen Wert vom Datentyp FILETIME zu einem Property hinzu.
AddHexBinary [▶ 126]	Fügt einen HexBinary-Wert zu einem Property hinzu.
AddKey [▶ 127]	Fügt einen neuen Property-Key an der aktuellen Position des SAX Writers hinzu.
AddKeyBool [▶ 127]	Legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp BOOL an.
AddKeyDateTime [▶ 128]	Legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp DATE_AND_TIME an.
AddKeyDcTime [▶ 128]	Legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp DCTIME an.
AddKeyFileTime [▶ 129]	Legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp FILETIME an.
AddKeyLreal [▶ 129]	Legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp LREAL an.
AddKeyNull [▶ 130]	Legt einen neuen Property-Key an und initialisiert dessen Wert mit null.
AddKeyNumber [▶ 130]	Legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp DINT an.
AddKeyString [▶ 131]	Legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp STRING an.
AddLint [▶ 131]	Fügt einen Wert vom Datentyp LINT zu einem Property hinzu.
AddLreal [▶ 132]	Fügt einen Wert vom Datentyp LREAL zu einem Property hinzu.
AddNull [▶ 132]	Fügt den Wert null zu einem Property hinzu.
AddRawArray [▶ 132]	Fügt ein gültiges JSON-Array zu einem gegebenen Property als Value hinzu.
AddRawObject [▶ 133]	Fügt ein gültiges JSON-Objekt zu einem gegebenen Property als Value hinzu.
AddReal [▶ 133]	Fügt einen Wert vom Datentyp REAL zu einem Property hinzu.
AddString [▶ 134]	Fügt einen Wert vom Datentyp STRING zu einem Property hinzu.
AddUdint [▶ 134]	Fügt einen Wert vom Datentyp UDINT zu einem Property hinzu.
AddUlint [▶ 135]	Fügt einen Wert vom Datentyp ULINT zu einem Property hinzu.
CopyDocument [▶ 135]	Kopiert den Inhalt des aktuell mit dem SAX Writer erstellten JSON-Objekts in eine Zielvariable vom Datentyp STRING.
EndArray [▶ 136]	Erzeugt den Abschluss eines angefangenen JSON-Arrays und fügt sie an der aktuellen Position des SAX Writers ein.
EndObject [▶ 136]	Erzeugt den Abschluss eines angefangenen JSON-Objekts und fügt sie an der aktuellen Position des SAX Writers ein.
GetDocument [▶ 137]	Liefert den Inhalt des aktuell mit dem SAX Writer erstellten JSON-Objekts und gibt diesen als Datentyp STRING(255) zurück.
GetDocumentLength [▶ 137]	Liefert die Länge des aktuell mit dem SAX Writer erstellten JSON-Objekts und gibt diese als Datentyp UDINT zurück.
GetMaxDecimalPlaces [▶ 138]	
IsComplete	
ResetDocument [▶ 138]	Setzt das aktuell mit dem SAX Writer erstellte JSON-Objekt zurück.
SetMaxDecimalPlaces [▶ 138]	

Name	Beschreibung
StartArray [▶ 139]	Erzeugt den Anfang eines neuen JSON-Arrays und fügt sie an der aktuellen Position des SAX Writers ein.
StartObject [▶ 139]	Erzeugt den Beginn eines neuen JSON-Objekts und fügt sie an der aktuellen Position des SAX Writers ein.

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\(\)](#) [[▶ 127](#)] ein entsprechendes Property erstellt.

Syntax

```

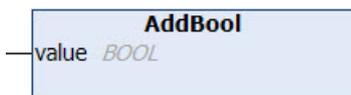
METHOD AddBase64
VAR_INPUT
    pBytes : Pointer TO BYTE;
    nBytes : DINT;
END_VAR

```

🔑 Eingänge

Name	Typ
pBytes	POINTER TO BYTE
nBytes	DINT

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\(\)](#) [[▶ 127](#)] ein entsprechendes Property erstellt.

Syntax

```

METHOD AddBool
VAR_INPUT
    value : BOOL;
END_VAR

```

🔑 Eingänge

Name	Typ
value	BOOL

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\(\)](#) [▶ 127] ein entsprechendes Property erstellt.

Syntax

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

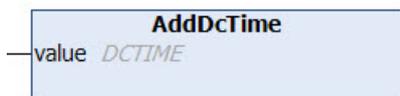
📌 Eingänge

Name	Typ
value	DATE_AND_TIME

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\(\)](#) [▶ 127] ein entsprechendes Property erstellt.

Syntax

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

📌 Eingänge

Name	Typ
value	DCTIME

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\(\)](#) [▶ 127] ein entsprechendes Property erstellt.

Syntax

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

Eingänge

Name	Typ
value	DINT

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\(\)](#) [\[► 127\]](#) ein entsprechendes Property erstellt.

Syntax

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

Eingänge

Name	Typ
value	FILETIME

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\(\)](#) [\[► 127\]](#) ein entsprechendes Property erstellt.

Syntax

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

 **Eingänge**

Name	Typ
pBytes	POINTER TO BYTE
nBytes	DINT

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 \[▶ 124\]](#), [AddBool \[▶ 124\]](#), [AddDateTime \[▶ 125\]](#), [AddDcTime \[▶ 125\]](#), [AddDint \[▶ 125\]](#), [AddFileTime \[▶ 126\]](#), [AddHexBinary \[▶ 126\]](#), [AddLint \[▶ 131\]](#), [AddLreal \[▶ 132\]](#), [AddNull \[▶ 132\]](#), [AddRawArray \[▶ 132\]](#), [AddRawObject \[▶ 133\]](#), [AddReal \[▶ 133\]](#), [AddString \[▶ 134\]](#), [AddUdint \[▶ 134\]](#), [AddUlint \[▶ 135\]](#).

Syntax

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

 **Ein-/Ausgänge**

Name	Typ
key	STRING

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
```

 **Eingänge**

Name	Typ
value	BOOL

🔌 / 📡 Ein-/Ausgänge

Name	Typ
key	STRING

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
```

🔌 Eingänge

Name	Typ
value	DATE_AND_TIME

🔌 / 📡 Ein-/Ausgänge

Name	Typ
key	STRING

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
```

 Eingänge

Name	Typ
value	DCTIME

 /  Ein-/Ausgänge

Name	Typ
key	STRING

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
```

 Eingänge

Name	Typ
value	FILETIME

 /  Ein-/Ausgänge

Name	Typ
key	STRING

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
```

Eingänge

Name	Typ
value	LREAL

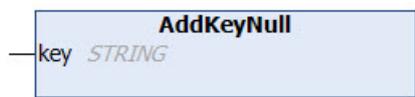
/ Ein-/Ausgänge

Name	Typ
key	STRING

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
```

/ Ein-/Ausgänge

Name	Typ
key	STRING

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
```

Eingänge

Name	Typ
value	DINT

 /  Ein-/Ausgänge

Name	Typ
key	STRING

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
```

 /  Ein-/Ausgänge

Name	Typ
key	STRING
value	STRING

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\(\)](#) [127] ein entsprechendes Property erstellt.

Syntax

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

 Eingänge

Name	Typ
value	LINT

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\(\)](#) [▶ 127] ein entsprechendes Property erstellt.

Syntax

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

📦 Eingänge

Name	Typ
value	LREAL

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\(\)](#) [▶ 127] 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\(\)](#) [▶ 127], [StartArray\(\)](#) [▶ 139] oder als erster Aufruf nach einem [ResetDocument\(\)](#) [▶ 138].

Syntax

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

 Rückgabewert

Name	Typ
AddRawArray	BOOL

 /  Ein-/Ausgänge

Name	Typ
rawJson	STRING

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\(\)](#) [▶ 127], [StartArray\(\)](#) [▶ 139] oder als erster Aufruf nach einem [ResetDocument\(\)](#) [▶ 138].

Syntax

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

 /  Ein-/Ausgänge

Name	Typ
rawJson	STRING

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\(\)](#) [▶ 127] ein entsprechendes Property erstellt.

Syntax

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

Eingänge

Name	Typ
value	REAL

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\(\)](#) [[127](#)] ein entsprechendes Property erstellt.

Syntax

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

Ein-/Ausgänge

Name	Typ
value	STRING

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\(\)](#) [[127](#)] ein entsprechendes Property erstellt.

Syntax

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

Eingänge

Name	Typ
value	UDINT

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\(\)](#) [► 127] ein entsprechendes Property erstellt.

Syntax

```

METHOD AddUlint
VAR_INPUT
    value : ULINT;
END_VAR
    
```

Eingänge

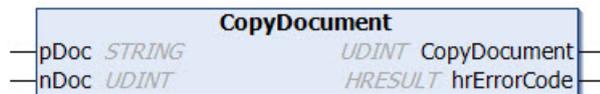
Name	Typ
value	ULINT

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
    
```

Rückgabewert

Name	Typ
CopyDocument	UDINT

Eingänge

Name	Typ
nDoc	UDINT

Ein-/Ausgänge

Name	Typ
pDoc	STRING

Ausgänge

Name	Typ
hrErrorCode	HRESULT

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
```

Rückgabewert

Name	Typ
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
```

Rückgabewert

Name	Typ
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 [▶ 135]()` verwendet werden.

Syntax

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

Rückgabewert

Name	Typ
GetDocument	STRING(255)

Ausgänge

Name	Typ
hrErrorCode	HRESULT

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
```

Rückgabewert

Name	Typ
GetDocumentLength	UDINT

Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

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

5.2.1.4.31 GetMaxDecimalPlaces

Diese Funktion fragt die aktuell eingestellte Anzahl an Nachkommastellen fest, nach denen eine Gleitkommazahl abgeschnitten wird.

GetMaxDecimalPlaces

DINT GetMaxDecimalPlaces

Syntax

```
METHOD GetMaxDecimalPlaces : DINT
```

Rückgabewert

Name	Typ	Beschreibung
GetMaxDecimalPlaces	DINT	Die aktuell eingestellte Anzahl an Stellen, nach der eine Gleitkommazahl abgeschnitten wird.

5.2.1.4.32 ResetDocument

ResetDocument

HRESULT ResetDocument

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

Syntax

```
METHOD ResetDocument : HRESULT
```

Rückgabewert

Name	Typ
ResetDocument	HRESULT

Beispielaufruf:

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

5.2.1.4.33 SetMaxDecimalPlaces

Diese Funktion legt die Anzahl an Nachkommastellen fest, nach denen eine Gleitkommazahl gerundet wird.

SetMaxDecimalPlaces

decimalPlaces DINT *HRESULT* SetMaxDecimalPlaces

Syntax

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

Rückgabewert

Name	Typ
SetMaxDecimalPlaces	HRESULT

 **Eingänge**

Name	Typ	Beschreibung
decimalPlaces	DINT	Die einzustellende Anzahl an Nachkommastellen, nach der eine Gleitkommazahl abgeschnitten wird.

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
```

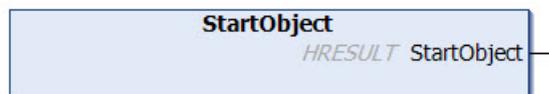
 **Rückgabewert**

Name	Typ
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
```

 **Rückgabewert**

Name	Typ
StartObject	HRESULT

Beispielaufruf:

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

5.2.1.5 FB_JsonSaxPrettyWriter



Dieser Funktionsbaustein hat einen Unterschied zum [FB_JsonSaxWriter](#) [121]: 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](#).

Syntax

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

Ausgänge

Name	Typ
initStatus	HRESULT

Methoden

Name	Beschreibung
SetFormatOptions [▶ 155]	Passt die allgemeinen Formatierungsoptionen für eine Instanz des [=>] FB_JsonSaxPrettyWriter an.
SetIndent [▶ 155]	Passt die Einrückung benutzerdefiniert an.

Alle weiteren Methoden finden Sie beim [FB_JsonSaxWriter](#) [▶ 121].

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\(\)](#) [▶ 143] ein entsprechendes Property erstellt.

Syntax

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

 Eingänge

Name	Typ
pBytes	POINTER TO BYTE
nBytes	DINT

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\(\)](#) [▶ 143] ein entsprechendes Property erstellt.

Syntax

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

 Eingänge

Name	Typ
value	BOOL

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\(\)](#) [▶ 143] ein entsprechendes Property erstellt.

Syntax

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

 Eingänge

Name	Typ
value	DATE_AND_TIME

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\(\)](#) [▶ 143] ein entsprechendes Property erstellt.

Syntax

```

METHOD AddDcTime
VAR_INPUT
  value : DCTIME;
END_VAR
  
```

📌 Eingänge

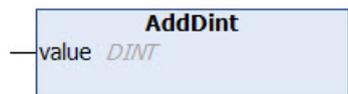
Name	Typ
value	DCTIME

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\(\)](#) [▶ 143] ein entsprechendes Property erstellt.

Syntax

```

METHOD AddDint
VAR_INPUT
  value : DINT;
END_VAR
  
```

📌 Eingänge

Name	Typ
value	DINT

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\(\)](#) [▶ 143] ein entsprechendes Property erstellt.

Syntax

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

 **Eingänge**

Name	Typ
value	FILETIME

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\(\)](#) [▶ 143] ein entsprechendes Property erstellt.

Syntax

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

 **Eingänge**

Name	Typ
pBytes	POINTER TO BYTE
nBytes	DINT

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](#) [▶ 140], [AddBool](#) [▶ 141], [AddDateTime](#) [▶ 141], [AddDcTime](#) [▶ 142], [AddDint](#) [▶ 142], [AddFileTime](#) [▶ 142], [AddHexBinary](#) [▶ 143], [AddLint](#) [▶ 148], [AddLreal](#) [▶ 148], [AddNull](#) [▶ 148], [AddRawArray](#) [▶ 149], [AddRawObject](#) [▶ 149], [AddReal](#) [▶ 150], [AddString](#) [▶ 150], [AddUdint](#) [▶ 151], [AddUlint](#) [▶ 151].

Syntax

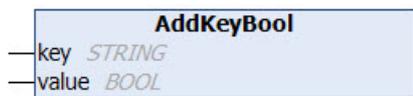
```
METHOD AddKey
VAR_IN_OUT CONSTANT
  key : STRING;
END_VAR
```

 **Ein-/Ausgänge**

Name	Typ
key	STRING

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
```

 **Eingänge**

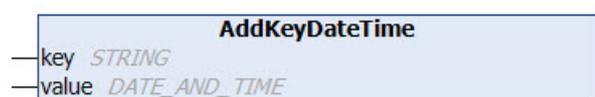
Name	Typ
value	BOOL

 **Ein-/Ausgänge**

Name	Typ
key	STRING

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
```

 **Eingänge**

Name	Typ
value	DATE_AND_TIME

 /  **Ein-/Ausgänge**

Name	Typ
key	STRING

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
```

 **Eingänge**

Name	Typ
value	DCTIME

 /  **Ein-/Ausgänge**

Name	Typ
key	STRING

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
```

🚩 Eingänge

Name	Typ
value	FILETIME

🚩 / 🚪 Ein-/Ausgänge

Name	Typ
key	STRING

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
```

🚩 Eingänge

Name	Typ
value	LREAL

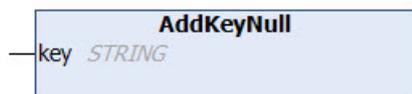
🚩 / 🚪 Ein-/Ausgänge

Name	Typ
key	STRING

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
```

 /  Ein-/Ausgänge

Name	Typ
key	STRING

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
```

 Eingänge

Name	Typ
value	DINT

 /  Ein-/Ausgänge

Name	Typ
key	STRING

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
```

 /  Ein-/Ausgänge

Name	Typ
key	STRING
value	STRING

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\(\)](#) [► 143] ein entsprechendes Property erstellt.

Syntax

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

🔗 Eingänge

Name	Typ
value	LINT

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\(\)](#) [► 143] ein entsprechendes Property erstellt.

Syntax

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

🔗 Eingänge

Name	Typ
value	LREAL

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\(\)](#) [▶ 143] 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\(\)](#) [▶ 143], [StartArray\(\)](#) [▶ 156] oder als erster Aufruf nach einem [ResetDocument\(\)](#) [▶ 154].

Syntax

METHOD AddRawArray
 VAR_IN_OUT CONSTANT
 rawJson : STRING;
 END_VAR

Rückgabewert

Name	Typ
AddRawArray	BOOL

Ein-/Ausgänge

Name	Typ
rawJson	STRING

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\(\)](#) [▶ 143], [StartArray\(\)](#) [▶ 156] oder als erster Aufruf nach einem [ResetDocument\(\)](#) [▶ 154].

Syntax

METHOD AddRawObject
 VAR_IN_OUT CONSTANT
 rawJson : STRING;
 END_VAR

Ein-/Ausgänge

Name	Typ
rawJson	STRING

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\(\)](#) [▶ 143] ein entsprechendes Property erstellt.

Syntax

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

Eingänge

Name	Typ
value	REAL

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\(\)](#) [▶ 143] ein entsprechendes Property erstellt.

Syntax

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

Ein-/Ausgänge

Name	Typ
value	STRING

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\(\)](#) [▶ 143] ein entsprechendes Property erstellt.

Syntax

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

🔌 Eingänge

Name	Typ
value	UDINT

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\(\)](#) [▶ 143] ein entsprechendes Property erstellt.

Syntax

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

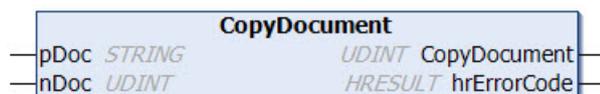
🔌 Eingänge

Name	Typ
value	ULINT

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

```

📦 Rückgabewert

Name	Typ
CopyDocument	UDINT

📦 Eingänge

Name	Typ
nDoc	UDINT

📦 / 📦 Ein-/Ausgänge

Name	Typ
pDoc	STRING

📦 Ausgänge

Name	Typ
hrErrorCode	HRESULT

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
```

📦 Rückgabewert

Name	Typ
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
```

Rückgabewert

Name	Typ
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 \[▶ 151\]\(\)](#) verwendet werden.

Syntax

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

Rückgabewert

Name	Typ
GetDocument	STRING(255)

Ausgänge

Name	Typ
hrErrorCode	HRESULT

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
```

Rückgabewert

Name	Typ
GetDocumentLength	UDINT

Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

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

5.2.1.5.31 GetMaxDecimalPlaces

Diese Funktion fragt die aktuell eingestellte Anzahl an Nachkommastellen fest, nach denen eine Gleitkommazahl abgeschnitten wird.



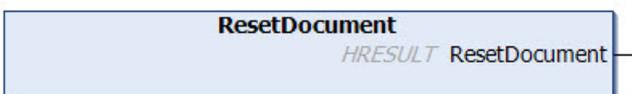
Syntax

```
METHOD GetMaxDecimalPlaces : DINT
```

Rückgabewert

Name	Typ	Beschreibung
GetMaxDecimalPlaces	DINT	Die aktuell eingestellte Anzahl an Stellen, nach der eine Gleitkommazahl abgeschnitten wird.

5.2.1.5.32 ResetDocument



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

Syntax

```
METHOD ResetDocument : HRESULT
```

Rückgabewert

Name	Typ
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 \[► 139\]](#) 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;
```

Rückgabewert

Name	Typ
SetFormatOptions	HRESULT

Eingänge

Name	Typ
options	EJsonPrettyFormatOptions

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
```

Rückgabewert

Name	Typ
SetIndent	HRESULT

Eingänge

Name	Typ
indentChar	SINT
indentCharCount	UDINT

Beispielaufruf:

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

5.2.1.5.35 SetMaxDecimalPlaces

Diese Funktion legt die Anzahl an Nachkommastellen fest, nach denen eine Gleitkommazahl gerundet wird.



Syntax

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

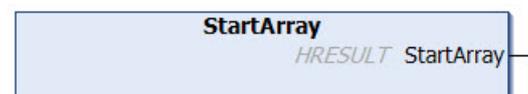
Rückgabewert

Name	Typ
SetMaxDecimalPlaces	HRESULT

Eingänge

Name	Typ	Beschreibung
decimalPlaces	DINT	Die einzustellende Anzahl an Nachkommastellen, nach der eine Gleitkommazahl abgeschnitten wird.

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
```

Rückgabewert

Name	Typ
StartArray	HRESULT

Beispielaufruf:

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

5.2.1.5.37 StartObject

StartObject
HRESULT 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

Rückgabewert

Name	Typ
StartObject	HRESULT

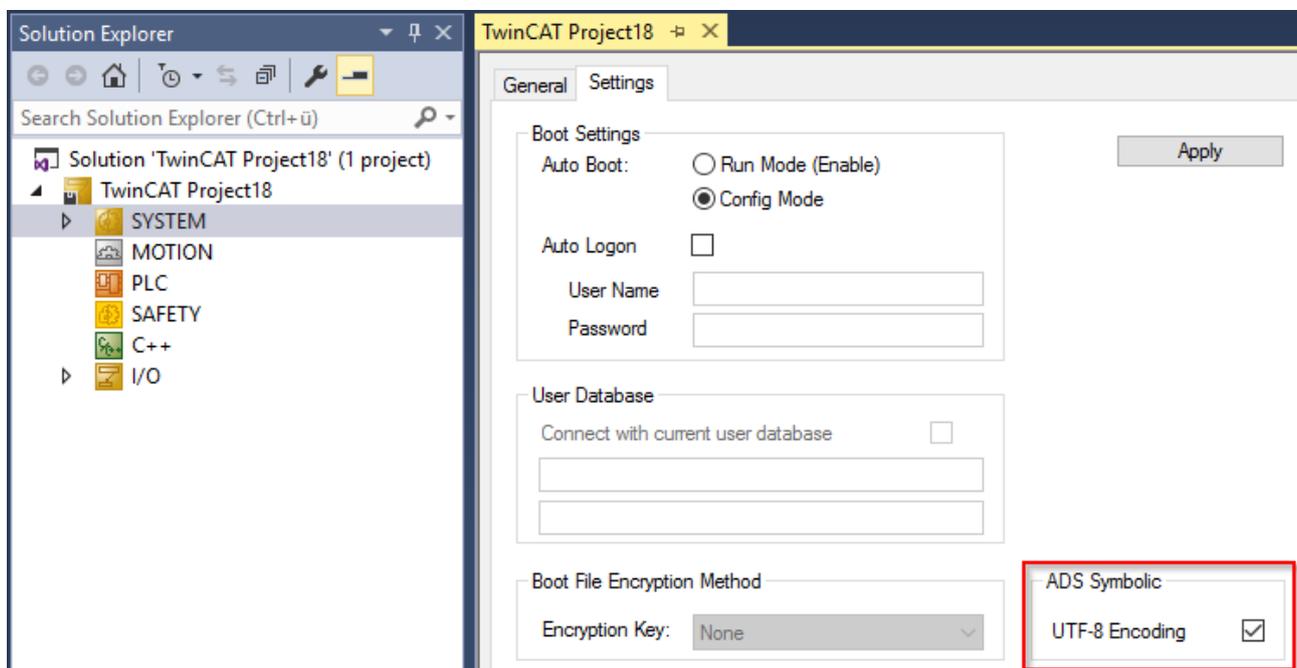
Beispielaufruf:

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

5.2.1.6 FB_JsonReadWriteDataType

FB_JsonReadWriteDatatype
HRESULT initStatus

Zur Verwendung von UTF-8-Zeichen, z. B. bei der automatischen Generierung von Metadaten über den Funktionsbaustein `FB_JsonReadWriteDataType` [▶ 157], 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

i 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](#).

🚪 Ausgänge

Name	Typ
initStatus	HRESULT

🔗 Methoden

Name	Beschreibung
FB_JsonReadWriteDataType	Metadaten werden über SPS-Attribute in die JSON-Repräsentation einer SPS-Datenstruktur auf einem FB_JsonSaxWriter-Objekt hinzugefügt.
AddJsonKeyValueFromSymbol [▶ 160]	Erzeugt die JSON-Repräsentation einer SPS-Datenstruktur auf einem FB_JsonSaxWriter-Objekt.
AddJsonValueFromSymbol [▶ 161]	Erzeugt die JSON-Repräsentation einer SPS-Datenstruktur auf einem FB_JsonSaxWriter-Objekt.
CopyJsonStringFromSymbol [▶ 162]	Erzeugt die JSON-Repräsentanz eines Symbols und kopiert diese in eine Variable vom Datentyp STRING.
CopyJsonStringFromSymbolProperties [▶ 163]	Erzeugt eine entsprechende JSON-Repräsentation von SPS-Attributen an einem Symbol.
CopySymbolNameByAddress [▶ 164]	Liefert den vollständigen (ADS-)Symbolnamen eines übergebenen Symbols.
GetDataTypeNameByAddress [▶ 165]	Liefert den Datentypnamen eines übergebenen Symbols.
GetJsonFromSymbol [▶ 165]	Erzeugt die entsprechende JSON-Repräsentation eines Symbols.
GetJsonStringFromSymbol [▶ 166]	Erzeugt die entsprechende JSON-Repräsentation eines Symbols.
GetJsonStringFromSymbolProperties [▶ 167]	Erzeugt eine entsprechende JSON-Repräsentation von SPS-Attributen an einem Symbol.
GetSizeJsonStringFromSymbol [▶ 168]	Liest die Größe der JSON-Repräsentanz eines Symbols aus.
GetSizeJsonStringFromSymbolProperties [▶ 169]	Liest die Größe der JSON-Repräsentanz von SPS-Attributen an einem Symbol aus.
GetSymbolNameByAddress [▶ 170]	Liefert den vollständigen (ADS-)Symbolnamen eines übergebenen Symbols.
SetSymbolFromJson [▶ 170]	Extrahiert einen String, der eine gültige JSON-Nachricht beinhaltet.

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` [▶ 121]-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
```

Rückgabewert

Name	Typ
AddJsonValueFromSymbol	BOOL

Ein-/Ausgänge

Name	Typ
fbWriter	FB_JsonSaxWriter
sKey	STRING
sDatatype	STRING
sProperties	STRING

Ausgänge

Name	Typ
hrErrorCode	HRESULT

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 AddJsonValueFromSymbol

AddJsonValueFromSymbol		
fbWriter	FB_JsonSaxWriter	BOOL AddJsonValueFromSymbol
sKey	STRING	HRESULT hrErrorCode
sDatatype	STRING	
nData	UDINT	
pData	PVOID	

Diese Methode erzeugt die JSON-Repräsentation einer SPS-Datenstruktur auf einem `FB_JsonSaxWriter` [▶ 121]-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()` [▶ 161] 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
```

Rückgabewert

Name	Typ
AddJsonValueFromSymbol	BOOL

Eingänge

Name	Typ
nData	UDINT
pData	PVOID

Ein-/Ausgänge

Name	Typ
fbWriter	FB_JsonSaxWriter
sKey	STRING
sDatatype	STRING

Ausgänge

Name	Typ
hrErrorCode	HRESULT

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

AddJsonValueFromSymbol		
fbWriter	FB_JsonSaxWriter	BOOL AddJsonValueFromSymbol
sDatatype	STRING	HRESULT hrErrorCode
nData	UDINT	
pData	PVOID	

Diese Methode erzeugt die JSON-Repräsentation einer SPS-Datenstruktur auf einem `FB_JsonSaxWriter` [► 121]-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
```

Rückgabewert

Name	Typ
AddJsonValueFromSymbol	BOOL

Eingänge

Name	Typ
nData	UDINT
pData	PVOID

Ein-/Ausgänge

Name	Typ
fbWriter	FB_JsonSaxWriter
sDatatype	STRING

Ausgänge

Name	Typ
hrErrorCode	HRESULT

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
```

Rückgabewert

Name	Typ
CopyJsonStringFromSymbol	UDINT

Eingänge

Name	Typ
nData	UDINT
nDoc	UDINT
pData	PVOID

Ein-/Ausgänge

Name	Typ
pDoc	STRING
sDatatype	STRING

Ausgänge

Name	Typ
hrErrorCode	HRESULT

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](#) [► 159] 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
```

Rückgabewert

Name	Typ
CopyJsonStringFromSymbolProperties	UDINT

Eingänge

Name	Typ
nDoc	UDINT

Ein-/Ausgänge

Name	Typ
pDoc	STRING
sDatatype	STRING
sProperties	STRING

Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

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

5.2.1.6.6 CopySymbolNameByAddress

CopySymbolNameByAddress	
nData	UDINT
pData	PVOID
sName	STRING
nName	UDINT

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
```

Rückgabewert

Name	Typ
CopySymbolNameByAddress	UDINT

Eingänge

Name	Typ
nData	UDINT
pData	PVOID
nName	UDINT

Ein-/Ausgänge

Name	Typ
sName	STRING

Ausgänge

Name	Typ
hrErrorCode	HRESULT

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.

● Strukturen mit einer einzelnen Variablen

i Wenn Strukturen mit einer einzelnen Variablen verwendet werden, kann über diese Methode der Datentyp nicht korrekt ausgelesen werden. Es kann über die Information von Größe und Adresse nicht zwischen der Struktur selbst und der einzelnen Variablen unterschieden werden.

Syntax

```

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

📌 Rückgabewert

Name	Typ
GetDataTypeNameByAddress	STRING

📌 Eingänge

Name	Typ
nData	UDINT
pData	PVOID

📌 Ausgänge

Name	Typ
hrErrorCode	HRESULT

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\(\)](#) [161] 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

```

🔴 Rückgabewert

Name	Typ
GetJsonFromSymbol	BOOL

🔴 Eingänge

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

🔴 / 🔴 Ein-/Ausgänge

Name	Typ
sDatatype	STRING

🔴 Ausgänge

Name	Typ
hrErrorCode	HRESULT

● Eingangsparameter nJson

i 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

GetJsonStringFromSymbol	
sDatatype <i>STRING</i>	<i>STRING(255)</i> GetJsonStringFromSymbol
nData <i>UDINT</i>	<i>HRESULT</i> hrErrorCode
pData <i>PVOID</i>	

Diese Methode erzeugt die entsprechende JSON-Repräsentation eines Symbols. Im Unterschied zur Methode [AddJsonValueFromSymbol\(\)](#) [161] 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 [▶ 162]()` 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
```

 **Rückgabewert**

Name	Typ
GetJsonStringFromSymbol	STRING(255)

 **Eingänge**

Name	Typ
nData	UDINT
pData	PVOID

 **Ein-/Ausgänge**

Name	Typ
sDatatype	STRING

 **Ausgänge**

Name	Typ
hrErrorCode	HRESULT

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 [▶ 159]` 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 [▶ 163]()` verwendet werden.

Syntax

```

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

```

🔑 Rückgabewert

Name	Typ
GetJsonStringFromSymbolProperties	STRING(255)

🔑 / 🔑 Ein-/Ausgänge

Name	Typ
sDatatype	STRING
sProperties	STRING

🔑 Ausgänge

Name	Typ
hrErrorCode	HRESULT

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

```

🔑 Rückgabewert

Name	Typ
GetSizeJsonStringFromSymbol	UDINT

 **Eingänge**

Name	Typ
nData	UDINT
pData	PVOID

 **Ein-/Ausgänge**

Name	Typ
sDatatype	STRING

 **Ausgänge**

Name	Typ
hrErrorCode	HRESULT

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
```

 **Rückgabewert**

Name	Typ
GetSizeJsonStringFromSymbolProperties	UDINT

 **Ein-/Ausgänge**

Name	Typ
sDatatype	STRING
sProperties	STRING

 **Ausgänge**

Name	Typ
hrErrorCode	HRESULT

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()` [► 164] verwendet werden.

Syntax

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

📌 Rückgabewert

Name	Typ
GetSymbolNameByAddress	STRING(255)

📌 Eingänge

Name	Typ
nData	UDINT
pData	PVOID

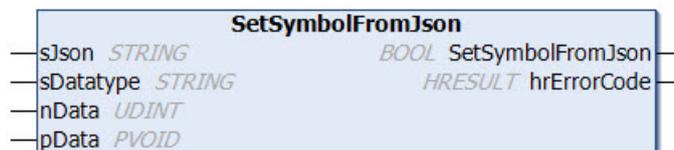
📌 Ausgänge

Name	Typ
hrErrorCode	HRESULT

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
```

 Rückgabewert

Name	Typ
SetSymbolFromJson	BOOL

 Eingänge

Name	Typ
nData	UDINT
pData	PVOID

 /  Ein-/Ausgänge

Name	Typ
sJson	STRING
sDatatype	STRING

 Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

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

5.2.1.7 FB_XmlDomParser



i 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](#).

 Ausgänge

Name	Typ
initStatus	HRESULT

☰ **Methoden**

Name	Beschreibung
AppendAttribute [▶ 175]	Fügt ein neues Attribut zu einem existierenden Knoten hinzu.
AppendAttributeAsBool [▶ 176]	Fügt ein neues Attribut zu einem existierenden Knoten hinzu (Boolean).
AppendAttributeAsDouble [▶ 177]	Fügt ein neues Attribut zu einem existierenden Knoten hinzu (Double).
AppendAttributeAsFloat [▶ 178]	Fügt ein neues Attribut zu einem existierenden Knoten hinzu (Float).
AppendAttributeAsInt [▶ 178]	Fügt ein neues Attribut zu einem existierenden Knoten hinzu (Integer).
AppendAttributeAsLint [▶ 179]	Fügt ein neues Attribut zu einem existierenden Knoten hinzu (Integer64).
AppendAttributeAsUInt [▶ 180]	Fügt ein neues Attribut zu einem existierenden Knoten hinzu (Unsigned Integer).
AppendAttributeAsUlint [▶ 181]	Fügt ein neues Attribut zu einem existierenden Knoten hinzu (Unsigned Integer64).
AppendAttributeCopy [▶ 181]	Fügt ein neues Attribut zu einem existierenden Knoten hinzu.
AppendChild [▶ 182]	Fügt einen neuen Knoten unterhalb eines existierenden Knotens ein.
AppendChildAsBool [▶ 183]	Fügt einen neuen Knoten unterhalb eines existierenden Knotens ein (Boolean).
AppendChildAsDouble [▶ 183]	Fügt einen neuen Knoten unterhalb eines existierenden Knotens ein (Double).
AppendChildAsFloat [▶ 184]	Fügt einen neuen Knoten unterhalb eines existierenden Knotens ein (Float).
AppendChildAsInt [▶ 185]	Fügt einen neuen Knoten unterhalb eines existierenden Knotens ein (Integer).
AppendChildAsLint [▶ 186]	Fügt einen neuen Knoten unterhalb eines existierenden Knotens ein (Integer64).
AppendChildAsUInt [▶ 186]	Fügt einen neuen Knoten unterhalb eines existierenden Knotens ein (Unsigned Integer).
AppendChildAsUlint [▶ 187]	Fügt einen neuen Knoten unterhalb eines existierenden Knotens ein (Unsigned Integer64).
AppendCopy [▶ 188]	Fügt einen neuen Knoten unterhalb eines existierenden Knotens ein.
AppendNode [▶ 188]	Fügt einen neuen Knoten unterhalb eines existierenden Knotens ein.
Attribute [▶ 189]	Liest das Attribut eines gegebenen XML-Knotens aus.
AttributeAsBool [▶ 190]	Liefert den Wert eines Attributs als Datentyp Boolean.
AttributeAsDouble [▶ 190]	Liefert den Wert eines Attributs als Datentyp Double.
AttributeAsFloat [▶ 191]	Liefert den Wert eines Attributs als Datentyp Float.
AttributeAsInt [▶ 191]	Liefert den Wert eines Attributs als Datentyp Integer.
AttributeAsLint [▶ 192]	Liefert den Wert eines Attributs als Datentyp Integer64.
AttributeAsUInt [▶ 192]	Liefert den Wert eines Attributs als Datentyp Unsigned Integer.
AttributeAsUlint [▶ 193]	Liefert den Wert eines Attributs als Datentyp Unsigned Integer64.
AttributeBegin [▶ 193]	Liefert einen Iterator über alle Attribute eines XML-Knotens.
AttributeFromIterator [▶ 194]	Konvertiert die aktuelle Position eines Iterators in ein XML-Attribut-Objekt.
AttributeName [▶ 194]	Liefert den Namen eines gegebenen Attributs.
Attributes [▶ 195]	Dient zur Navigation durch den DOM und liefert einen Iterator auf alle gefundenen Attribute an einem XML-Knoten zurück.
AttributeText [▶ 195]	Liefert den Text eines gegebenen Attributs.
Begin [▶ 196]	Liefert einen Iterator über alle Kindelemente eines XML-Knotens.
BeginByName [▶ 196]	Liefert einen Iterator über alle Kindelemente eines XML-Knotens, beginnend ab einem bestimmten Element.
Child [▶ 197]	Dient zur Navigation durch den DOM und liefert eine Referenz auf das (erste) Kindelement des aktuellen Knotens zurück.

Name	Beschreibung
ChildByAttribute [▶ 197]	Dient zur Navigation durch den DOM und liefert eine Referenz auf ein Kindelement im XML-Dokument zurück.
ChildByAttributeAndName [▶ 198]	Dient zur Navigation durch den DOM und liefert eine Referenz auf ein Kindelement im XML-Dokument zurück.
ChildByName [▶ 199]	Dient zur Navigation durch den DOM und liefert eine Referenz auf ein Kindelement im XML-Dokument zurück.
Children [▶ 199]	Dient zur Navigation durch den DOM und liefert einen Iterator auf mehrere gefundene Kindelemente im XML-Dokument zurück.
ChildrenByName [▶ 200]	Dient zur Navigation durch den DOM und liefert einen Iterator auf mehrere gefundene Kindelemente im XML-Dokument zurück.
ClearIterator [▶ 201]	Diese Methode setzt bereits verwendete Iteratoren zurück, sodass diese bei dem nächsten Programmdurchlauf wieder verwendet werden können.
Compare [▶ 202]	Überprüft zwei Iteratoren auf Gleichheit.
CopyAttributeText [▶ 202]	Liest den Wert eines XML-Attributs aus und schreibt diesen in eine Variable vom Datentyp String.
CopyDocument [▶ 203]	Kopiert den Inhalt des DOM-Speichers in eine Variable vom Datentyp String.
CopyNodeText [▶ 204]	Liest den Wert eines XML-Knotens aus und schreibt diesen in eine Variable vom Datentyp String.
CopyNodeXml [▶ 204]	Liest die XML-Struktur eines XML-Knotens aus und schreibt diese in eine Variable vom Datentyp String.
End	
FirstNodeByPath [▶ 205]	Navigiert anhand eines an die Methode übergebenen Pfads durch ein XML-Dokument.
GetAttributeTextLength [▶ 206]	Liefert die Länge des Werts eines XML-Attributs.
GetDocumentLength [▶ 206]	Liefert die Länge eines XML-Dokuments in Bytes.
GetDocumentNode [▶ 206]	Liefert den Root-Knoten eines XML-Dokuments.
GetNodeTextLength [▶ 207]	Liefert die Länge des Werts eines XML-Knotens.
GetNodeXmlLength [▶ 207]	Liefert die Länge der XML-Struktur eines XML-Knotens.
GetRootNode [▶ 208]	Liefert eine Referenz zum ersten XML-Knoten im XML-Dokument.
InsertAttributeCopy [▶ 208]	Fügt ein Attribut zu einem XML-Knoten hinzu und kopiert hierbei den Namen und den Wert eines existierenden Attributs.
InsertAttribute [▶ 209]	Fügt ein Attribut zu einem XML-Knoten hinzu.
InsertChild [▶ 209]	Fügt einen Knoten zu einem existierenden XML-Knoten hinzu.
InsertCopy [▶ 210]	Fügt einen neuen Knoten zu einem existierenden XML Knoten hinzu und kopiert hierbei einen existierenden Knoten.
IsEnd [▶ 211]	Überprüft, ob sich ein gegebener XML-Iterator am Ende der zu durchlaufenden Iteration befindet.
LoadDocumentFromFile [▶ 211]	Lädt ein XML-Dokument aus einer Datei.
NewDocument [▶ 212]	Erzeugt ein leeres XML-Dokument im DOM-Speicher.
Next [▶ 213]	Setzt einen XML-Iterator auf das nächste zu durchlaufende Objekt.
NextAttribute [▶ 213]	Liefert zu einem gegebenen XML-Attribut das nächste Attribut.
NextByName [▶ 214]	Setzt einen XML-Iterator auf das nächste zu durchlaufende Objekt, das anhand seines Namens identifiziert wird.
NextSibling [▶ 214]	Liefert zu einem gegebenen XML-Knoten den nächsten direkten Knoten auf derselben XML-Ebene.
NextSiblingByName [▶ 215]	Liefert zu einem gegebenen XML-Knoten den nächsten direkten Knoten mit einem bestimmten Namen auf derselben XML-Ebene.
Node [▶ 216]	Wird in Zusammenhang mit einem Iterator verwendet, um durch den DOM zu navigieren.

Name	Beschreibung
NodeAsBool [▶ 216]	Liefert den Text eines XML-Knotens als Datentyp Boolean.
NodeAsDouble [▶ 217]	Liefert den Text eines XML-Knotens als Datentyp Double.
NodeAsFloat [▶ 217]	Liefert den Text eines XML-Knotens als Datentyp Float.
NodeAsInt [▶ 218]	Liefert den Text eines XML-Knotens als Datentyp Integer.
NodeAsLint [▶ 218]	Liefert den Text eines XML-Knotens als Datentyp Integer64.
NodeAsUInt [▶ 219]	Liefert den Text eines XML-Knotens als Datentyp Unsigned Integer.
NodeAsUlint [▶ 219]	Liefert den Text eines XML-Knotens als Datentyp Unsigned Integer64.
NodeName [▶ 220]	Gibt den Namen eines XML-Knotens zurück.
NodeText [▶ 220]	Liefert den Text eines XML-Knotens.
ParseDocument [▶ 221]	Lädt ein XML-Dokument zur weiteren Verarbeitung in den DOM-Speicher.
RemoveAttribute	
RemoveAttributeByName	
RemoveChild [▶ 221]	entfernt einen XML-Kindknoten von einem gegebenen XML-Knoten.
RemoveChildByName [▶ 222]	Entfernt einen XML-Kindknoten von einem gegebenen XML-Knoten.
SaveDocumentToFile [▶ 222]	Speichert das aktuelle XML-Dokument in einer Datei.
SetAttribute [▶ 223]	Setzt den Wert eines Attributs.
SetAttributeAsBool [▶ 224]	Setzt den Wert eines Attributs (Boolean).
SetAttributeAsDouble [▶ 225]	Setzt den Wert eines Attributs (Double).
SetAttributeAsFloat [▶ 225]	Setzt den Wert eines Attributs (Float).
SetAttributeAsInt [▶ 226]	Setzt den Wert eines Attributs (Integer).
SetAttributeAsLint [▶ 226]	Setzt den Wert eines Attributs (Integer64).
SetAttributeAsUInt [▶ 227]	Setzt den Wert eines Attributs (Unsigned Integer).
SetAttributeAsUlint [▶ 227]	Setzt den Wert eines Attributs (Unsigned Integer64).
SetChild [▶ 228]	Setzt den Wert eines XML-Knotens (String).
SetChildAsBool [▶ 228]	Setzt den Wert eines XML-Knotens (Boolean).
SetChildAsDouble [▶ 229]	Setzt den Wert eines XML-Knotens (Double).
SetChildAsFloat [▶ 229]	Setzt den Wert eines XML-Knotens (Float).
SetChildAsInt [▶ 230]	Setzt den Wert eines XML-Knotens (Integer).
SetChildAsLint [▶ 230]	Setzt den Wert eines XML-Knotens (Integer64).
SetChildAsUInt [▶ 231]	Setzt den Wert eines XML-Knotens (Unsigned Integer).
SetChildAsUlint [▶ 232]	Setzt den Wert eines XML-Knotens (Unsigned Integer64).

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

```

📌 Rückgabewert

Name	Typ
AppendAttribute	SXmlAttribute

📌 Eingänge

Name	Typ
n	SXmlNode

📌 / 📌 Ein-/Ausgänge

Name	Typ
name	STRING
value	STRING

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

```

📌 Rückgabewert

Name	Typ
AppendAttributeAsBool	SXmlAttribute

 **Eingänge**

Name	Typ
n	SXmlNode
value	BOOL

 **Ein-/Ausgänge**

Name	Typ
name	STRING

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
```

 **Rückgabewert**

Name	Typ
AppendAttributeAsDouble	SXmlAttribute

 **Eingänge**

Name	Typ
n	SXmlNode
value	LREAL

 **Ein-/Ausgänge**

Name	Typ
name	STRING

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
```

Rückgabewert

Name	Typ
AppendAttributeAsFloat	SXmlAttribute

Eingänge

Name	Typ
n	SXmlNode
value	REAL

Ein-/Ausgänge

Name	Typ
name	STRING

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
```

 Rückgabewert

Name	Typ
AppendAttributeAsInt	SXmlAttribute

 Eingänge

Name	Typ
n	SXmlNode
value	DINT

 /  Ein-/Ausgänge

Name	Typ
name	STRING

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
```

 Rückgabewert

Name	Typ
AppendAttributeAsLint	SXmlAttribute

Eingänge

Name	Typ
n	SXmlNode
value	LINT

/ Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

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

5.2.1.7.7 AppendAttributeAsUint

AppendAttributeAsUint	
n	SXmlNode
name	STRING
value	UDINT

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
```

Rückgabewert

Name	Typ
AppendAttributeAsUint	SXmlAttribute

Eingänge

Name	Typ
n	SXmlNode
value	UDINT

/ Ein-/Ausgänge

Name	Typ
name	STRING

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
    
```

Rückgabewert

Name	Typ
AppendAttributeAsUlint	SXmlAttribute

Eingänge

Name	Typ
n	SXmlNode
value	ULINT

Ein-/Ausgänge

Name	Typ
name	STRING

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
    
```

Rückgabewert

Name	Typ
AppendAttributeCopy	SXmlAttribute

Eingänge

Name	Typ
n	SXmlNode
copy	SXmlAttribute

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
  
```

Rückgabewert

Name	Typ
AppendChild	SXmlNode

Eingänge

Name	Typ
n	SXmlNode
cdata	BOOL

Ein-/Ausgänge

Name	Typ
name	STRING
value	STRING

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
```

Rückgabewert

Name	Typ
AppendChildAsBool	SXmlNode

Eingänge

Name	Typ
n	SXmlNode
value	BOOL

Ein-/Ausgänge

Name	Typ
name	STRING

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

```

🔑 Rückgabewert

Name	Typ
AppendChildAsDouble	SXmlNode

🔑 Eingänge

Name	Typ
n	SXmlNode
value	LREAL

🔑 / 🔑 Ein-/Ausgänge

Name	Typ
name	STRING

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

```

🔑 Rückgabewert

Name	Typ
AppendChildAsFloat	SXmlNode

 **Eingänge**

Name	Typ
n	SXmlNode
value	REAL

 /  **Ein-/Ausgänge**

Name	Typ
name	STRING

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
```

 **Rückgabewert**

Name	Typ
AppendChildAsInt	SXmlNode

 **Eingänge**

Name	Typ
n	SXmlNode
value	DINT

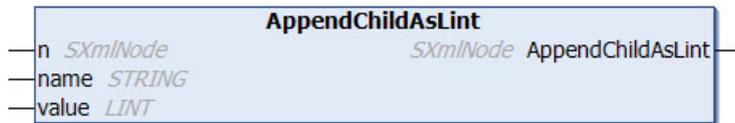
 /  **Ein-/Ausgänge**

Name	Typ
name	STRING

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
```

Rückgabewert

Name	Typ
AppendChildAsLint	SXmlNode

Eingänge

Name	Typ
n	SXmlNode
value	LINT

Ein-/Ausgänge

Name	Typ
name	STRING

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
```

```
END_VAR
VAR_INPUT
  value : UDINT;
END_VAR
```

 Rückgabewert

Name	Typ
AppendChildAsUint	SXmlNode

 Eingänge

Name	Typ
n	SXmlNode
value	UDINT

 /  Ein-/Ausgänge

Name	Typ
name	STRING

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
```

 Rückgabewert

Name	Typ
AppendChildAsUlint	SXmlNode

 Eingänge

Name	Typ
n	SXmlNode
value	ULINT

📁 / 📁 Ein-/Ausgänge

Name	Typ
name	STRING

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
  
```

📁 Rückgabewert

Name	Typ
AppendCopy	SXmlNode

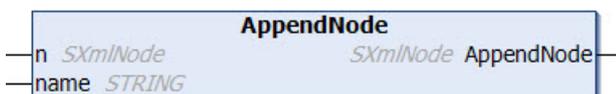
📁 Eingänge

Name	Typ
n	SXmlNode
copy	SXmlNode

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
  
```

 Rückgabewert

Name	Typ
AppendNode	SXmlNode

 Eingänge

Name	Typ
n	SXmlNode

 /  Ein-/Ausgänge

Name	Typ
name	STRING

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
```

 Rückgabewert

Name	Typ
Attribute	SXmlAttribute

 Eingänge

Name	Typ
n	SXmlNode

 /  Ein-/Ausgänge

Name	Typ
name	STRING

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
```

Rückgabewert

Name	Typ
AttributeAsBool	BOOL

Eingänge

Name	Typ
a	SXmlAttribute

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
```

Rückgabewert

Name	Typ
AttributeAsDouble	LREAL

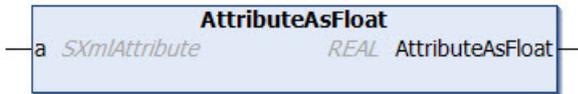
Eingänge

Name	Typ
a	SXmlAttribute

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
```

Rückgabewert

Name	Typ
AttributeAsFloat	REAL

Eingänge

Name	Typ
a	SXmlAttribute

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
```

Rückgabewert

Name	Typ
AttributeAsInt	DINT

Eingänge

Name	Typ
a	SXmlAttribute

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
```

Rückgabewert

Name	Typ
AttributeAsLint	LINT

Eingänge

Name	Typ
a	SXmlAttribute

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
```

Rückgabewert

Name	Typ
AttributeAsUint	UDINT

Eingänge

Name	Typ
a	SXmlAttribute

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
```

Rückgabewert

Name	Typ
AttributeAsUlint	ULINT

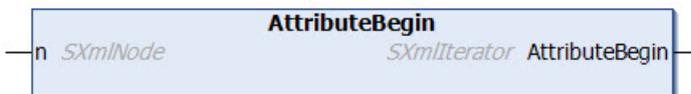
Eingänge

Name	Typ
a	SXmlAttribute

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
```

Rückgabewert

Name	Typ
AttributeBegin	SXmlIterator

Eingänge

Name	Typ
n	SXmlNode

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
```

Rückgabewert

Name	Typ
AttributeFromIterator	SXmlAttribute

Eingänge

Name	Typ
it	SXmlIterator

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
```

Rückgabewert

Name	Typ
AttributeName	STRING

Eingänge

Name	Typ
a	SXmlAttribute

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 : SXmlNodeAttribute
VAR_INPUT
  n : SXmlNode;
  it : REFERENCE TO SXmlNodeIterator;
END_VAR
```

Rückgabewert

Name	Typ
Attributes	SXmlNodeAttribute

Eingänge

Name	Typ
it	REFERENCE TO SXmlNodeIterator

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 : SXmlNodeAttribute;
END_VAR
```

Rückgabewert

Name	Typ
AttributeText	STRING(255)

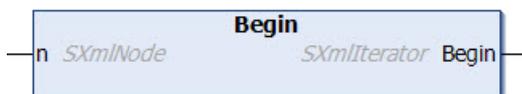
Eingänge

Name	Typ
a	SXmlNodeAttribute

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 : SXmlNodeIterator
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Rückgabewert

Name	Typ
Begin	SXmlNodeIterator

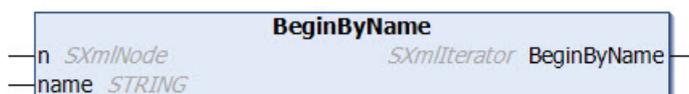
Eingänge

Name	Typ
n	SXmlNode

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 : SXmlNodeIterator
VAR_INPUT
  n : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name : STRING;
END_VAR
```

Rückgabewert

Name	Typ
BeginByName	SXmlNodeIterator

 **Eingänge**

Name	Typ
n	SXmlNode

 **Ein-/Ausgänge**

Name	Typ
name	STRING

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
```

 **Rückgabewert**

Name	Typ

 **Eingänge**

Name	Typ
n	SXmlNode

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

```

🔑 Rückgabewert

Name	Typ
ChildByAttribute	SXmlNode

🔑 Eingänge

Name	Typ
n	SXmlNode

🔑 / 🔑 Ein-/Ausgänge

Name	Typ
attr	STRING
value	STRING

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

```

🔑 Rückgabewert

Name	Typ
ChildByAttributeAndName	SXmlNode

 **Eingänge**

Name	Typ
n	SXmlNode

 /  **Ein-/Ausgänge**

Name	Typ
attr	STRING
value	STRING
child	STRING

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
```

 **Rückgabewert**

Name	Typ
ChildByName	SXmlNode

 **Eingänge**

Name	Typ
n	SXmlNode

 /  **Ein-/Ausgänge**

Name	Typ
name	STRING

Beispielaufruf:

```
xmlMachines := fbXml.Children(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
```

Rückgabewert

Name	Typ
Children	SXmlNode

Eingänge

Name	Typ
n	SXmlNode

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
```

Rückgabewert

Name	Typ
ChildrenByName	SXmlNode

 **Eingänge**

Name	Typ
n	SXmlNode
it	REFERENCE TO SXmlIterator

 **Ein-/Ausgänge**

Name	Typ
name	STRING

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 ClearIterator



Diese Methode setzt bereits verwendete Iteratoren zurück, sodass diese bei dem nächsten Programmdurchlauf wieder verwendet werden können.

Syntax

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

 **Rückgabewert**

Name	Typ
ClearIterator	BOOL

 **Eingänge**

Name	Typ
it	SXmlIterator

Beispielaufruf:

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

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4024.56	x86, x64, Arm®	Tc3_JsonXml 3.4.5.0

5.2.1.7.42 Compare



Diese Methode überprüft zwei Iteratoren auf Gleichheit.

Syntax

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

Rückgabewert

Name	Typ
Compare	BOOL

Eingänge

Name	Typ
It1	SXmlIterator
It2	SXmlIterator

Beispielaufruf:

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

5.2.1.7.43 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
```

Rückgabewert

Name	Typ
CopyAttributeText	UDINT

 **Eingänge**

Name	Typ
a	SXmlAttribute
nXml	UDINT

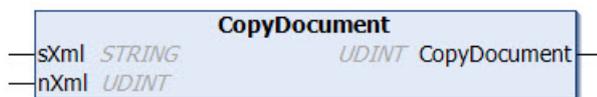
 /  **Ein-/Ausgänge**

Name	Typ
sXml	STRING

Beispielaufruf:

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

5.2.1.7.44 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
```

 **Rückgabewert**

Name	Typ
CopyDocument	UDINT

 **Eingänge**

Name	Typ
nXml	UDINT

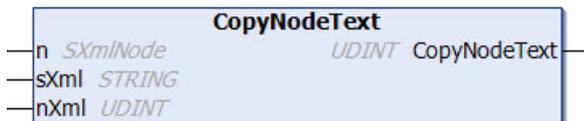
 /  **Ein-/Ausgänge**

Name	Typ
sXml	STRING

Beispielaufruf:

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

5.2.1.7.45 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
```

Rückgabewert

Name	Typ
CopyNodeText	UDINT

Eingänge

Name	Typ
n	SXmlNode
nXml	UDINT

Ein-/Ausgänge

Name	Typ
sXml	STRING

Beispielaufruf:

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

5.2.1.7.46 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
```

 Rückgabewert

Name	Typ
CopyNodeXml	UDINT

 Eingänge

Name	Typ
a	SXmlNode
nXml	UDINT

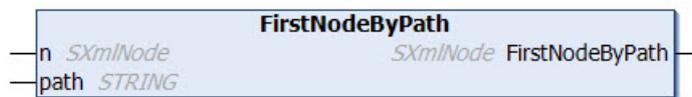
 /  Ein-/Ausgänge

Name	Typ
sXml	STRING

Beispielaufruf:

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

5.2.1.7.47 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
```

 Rückgabewert

Name	Typ
FirstNodeByPath	SXmlNode

 Eingänge

Name	Typ
n	SXmlNode
path	STRING

Beispielaufruf:

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

5.2.1.7.48 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
```

Rückgabewert

Name	Typ
GetAttributeTextLength	UDINT

Eingänge

Name	Typ
a	SXmlAttribute

Beispielaufruf:

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

5.2.1.7.49 GetDocumentLength



Diese Methode liefert die Länge eines XML-Dokuments in Bytes.

Syntax

```
METHOD GetDocumentLength : UDINT
```

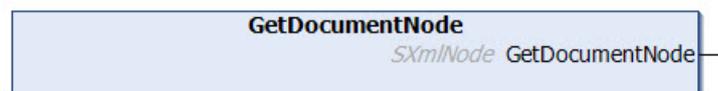
Rückgabewert

Name	Typ
GetDocumentLength	UDINT

Beispielaufruf:

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

5.2.1.7.50 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
```

 **Rückgabewert**

Name	Typ
GetDocumentNode	SXmlNode

Beispielaufruf:

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

5.2.1.7.51 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
```

 **Rückgabewert**

Name	Typ
GetNodeTextLength	UDINT

 **Eingänge**

Name	Typ
n	SXmlNode

Beispielaufruf:

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

5.2.1.7.52 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
```

🔪 Rückgabewert

Name	Typ
GetNodeXmlLength	UDINT

🔪 Eingänge

Name	Typ
n	SXmlNode

Beispielaufruf:

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

5.2.1.7.53 GetRootNode



Diese Methode liefert eine Referenz zum ersten XML-Knoten im XML-Dokument.

Syntax

```
METHOD GetRootNode : SXmlNode
```

🔪 Rückgabewert

Name	Typ
GetRootNode	SXmlNode

Beispielaufruf:

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

5.2.1.7.54 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
```

🔪 Rückgabewert

Name	Typ
InsertAttributeCopy	SXmlAttribute

 **Eingänge**

Name	Typ
n	SXmlNode
before	SXmlAttribute
copy	SXmlAttribute

Beispielaufruf:

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

5.2.1.7.55 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
```

 **Rückgabewert**

Name	Typ
InsertAttribute	SXmlAttribute

 **Eingänge**

Name	Typ
n	SXmlNode
before	SXmlAttribute

 **Ein-/Ausgänge**

Name	Typ
name	STRING

Beispielaufruf:

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

5.2.1.7.56 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
```

Rückgabewert

Name	Typ
InsertChild	SXmlNode

Eingänge

Name	Typ
n	SXmlNode
before	SXmlAttribute

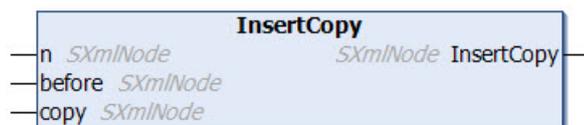
Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

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

5.2.1.7.57 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
```

➡ Rückgabewert

Name	Typ
InsertCopy	SXmlNode

➡ Eingänge

Name	Typ
n	SXmlNode
before	SXmlNode
copy	SXmlNode

Beispielaufruf:

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

5.2.1.7.58 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
```

➡ Rückgabewert

Name	Typ
IsEnd	BOOL

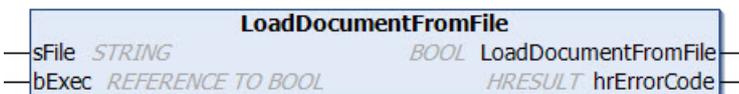
➡ Eingänge

Name	Typ
nit	SXmlIterator

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
```

5.2.1.7.59 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.

● Encoding



Bitte beachten Sie, dass die eingelesenen Dateien in UTF-8 formatiert sein müssen. Formate wie bspw. UTF-16 werden nicht unterstützt.

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
```

📌 Rückgabewert

Name	Typ
LoadDocumentFromFile	BOOL

📌 Eingänge

Name	Typ
bExec	REFERENCE TO BOOL

📌 / 📌 Ein-/Ausgänge

Name	Typ
sFile	STRING

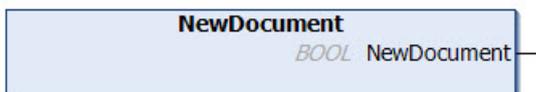
📌 Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

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

5.2.1.7.60 NewDocument



Diese Methode erzeugt ein leeres XML-Dokument im DOM-Speicher.

Syntax

```
METHOD NewDocument : BOOL
```

Rückgabewert

Name	Typ
NewDocument	BOOL

Beispielaufruf:

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

5.2.1.7.61 Next



Diese Methode setzt einen XML-Iterator auf das nächste zu durchlaufende Objekt.

Syntax

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

Rückgabewert

Name	Typ
Next	SXmlIterator

Eingänge

Name	Typ
it	SXmlIterator

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
```

5.2.1.7.62 NextAttribute



Diese Methode liefert zu einem gegebenen XML-Attribut das nächste Attribut.

Syntax

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

Rückgabewert

Name	Typ
NextAttribute	SXmlAttribute

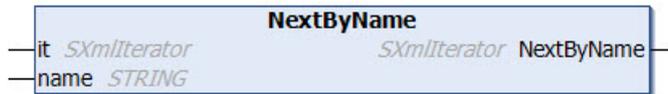
Eingänge

Name	Typ
a	SXmlAttribute

Beispielaufruf:

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

5.2.1.7.63 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

```

Rückgabewert

Name	Typ
NextByName	SXmlIterator

Eingänge

Name	Typ
it	SXmlIterator

Ein-/Ausgänge

Name	Typ
name	STRING

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

```

5.2.1.7.64 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
```

 **Rückgabewert**

Name	Typ
NextSibling	SXmlNode

 **Eingänge**

Name	Typ
n	SXmlNode

Beispielaufruf:

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

5.2.1.7.65 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
```

 **Rückgabewert**

Name	Typ
NextSiblingByName	SXmlNode

 **Eingänge**

Name	Typ
n	SXmlNode

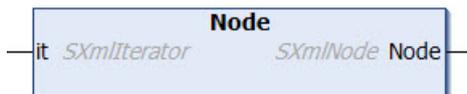
 **Ein-/Ausgänge**

Name	Typ
name	STRING

Beispielaufruf:

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

5.2.1.7.66 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
  
```

Rückgabewert

Name	Typ
Node	SXmlNode

Eingänge

Name	Typ
it	SXmlIterator

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
  
```

5.2.1.7.67 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
  
```

Rückgabewert

Name	Typ
NodeAsBool	BOOL

Eingänge

Name	Typ
n	SXmlNode

Beispielaufruf:

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

5.2.1.7.68 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
```

Rückgabewert

Name	Typ
NodeAsDouble	LREAL

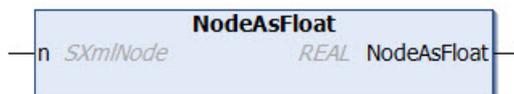
Eingänge

Name	Typ
n	SXmlNode

Beispielaufruf:

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

5.2.1.7.69 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
```

Rückgabewert

Name	Typ
NodeAsFloat	REAL

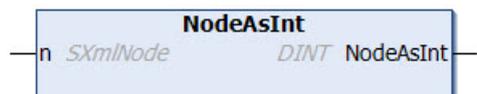
Eingänge

Name	Typ
n	SXmlNode

Beispielaufruf:

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

5.2.1.7.70 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
```

Rückgabewert

Name	Typ
NodeAsInt	DINT

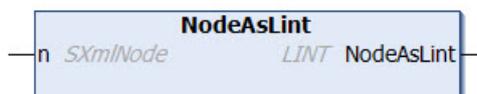
Eingänge

Name	Typ
n	SXmlNode

Beispielaufruf:

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

5.2.1.7.71 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
```

Rückgabewert

Name	Typ
NodeAsLint	LINT

Eingänge

Name	Typ
n	SXmlNode

Beispielaufruf:

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

5.2.1.7.72 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
```

Rückgabewert

Name	Typ
NodeAsUint	UDINT

Eingänge

Name	Typ
n	SXmlNode

Beispielaufruf:

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

5.2.1.7.73 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
```

Rückgabewert

Name	Typ
NodeAsUlint	ULINT

Eingänge

Name	Typ
n	SXmlNode

Beispielaufruf:

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

5.2.1.7.74 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
  
```

Rückgabewert

Name	Typ
nodeName	STRING

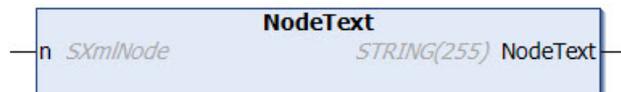
Eingänge

Name	Typ
n	SXmlNode

Beispielaufruf:

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

5.2.1.7.75 NodeText



Diese Methode liefert den Text eines XML-Knotens. Der XML-Knoten wird der Methode als Eingangsparameter übergeben.

Wenn der Text eines XML-Knotens länger als 255 Zeichen ist, muss die Methode [CopyNodeText](#) ID 2041 verwendet werden.

Syntax

```

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

Rückgabewert

Name	Typ
NodeText	STRING(255)

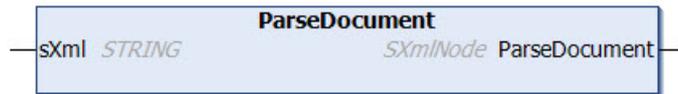
Eingänge

Name	Typ
n	SXmlNode

Beispielaufruf:

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

5.2.1.7.76 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
```

Rückgabewert

Name	Typ
ParseDocument	SXmlNode

Ein-/Ausgänge

Name	Typ
sXml	STRING

Beispielaufruf:

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

5.2.1.7.77 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
```

Rückgabewert

Name	Typ
RemoveChild	BOOL

Eingänge

Name	Typ
n	SXmlNode
child	SXmlNode

Beispielaufruf:

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

5.2.1.7.78 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
```

Rückgabewert

Name	Typ
RemoveChildByName	BOOL

Eingänge

Name	Typ
n	SXmlNode

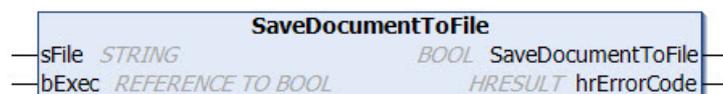
Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

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

5.2.1.7.79 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
```

 Rückgabewert

Name	Typ
SaveDocumentToFile	BOOL

 Eingänge

Name	Typ
bExec	REFERENCE TO BOOL

 /  Ein-/Ausgänge

Name	Typ
sFile	STRING

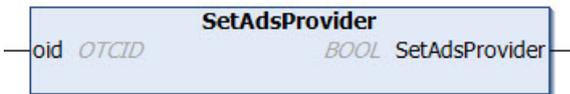
 Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

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

5.2.1.7.80 SetAdsProvider



Syntax

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

 Rückgabewert

Name	Typ
SetAdsProvider	BOOL

 /  Ein-/Ausgänge

Name	Typ
oid	OTCID

5.2.1.7.81 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
```

Rückgabewert

Name	Typ
SetAttribute	SXmlAttribute

Eingänge

Name	Typ
a	SXmlAttribute

Ein-/Ausgänge

Name	Typ
value	STRING

Beispielaufruf:

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

5.2.1.7.82 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
```

Rückgabewert

Name	Typ
SetAttributeAsBool	SXmlAttribute

Eingänge

Name	Typ
a	SXmlAttribute
value	BOOL

Beispielaufruf:

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

5.2.1.7.83 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
```

Rückgabewert

Name	Typ
SetAttributeAsDouble	SXmlAttribute

Eingänge

Name	Typ
a	SXmlAttribute
value	LREAL

Beispielaufruf:

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

5.2.1.7.84 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
```

Rückgabewert

Name	Typ
SetAttributeAsFloat	SXmlAttribute

Eingänge

Name	Typ
a	SXmlAttribute
value	REAL

Beispielaufruf:

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

5.2.1.7.85 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
```

Rückgabewert

Name	Typ
SetAttributeAsInt	SXmlAttribute

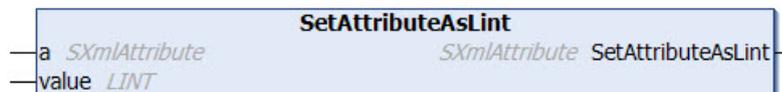
Eingänge

Name	Typ
a	SXmlAttribute
value	DINT

Beispielaufruf:

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

5.2.1.7.86 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
```

Rückgabewert

Name	Typ
SetAttributeAsLint	SXmlAttribute

Eingänge

Name	Typ
a	SXmlAttribute
value	LINT

Beispielaufruf:

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

5.2.1.7.87 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
```

Rückgabewert

Name	Typ
SetAttributeAsUint	SXmlAttribute

Eingänge

Name	Typ
a	SXmlAttribute
value	UDINT

Beispielaufruf:

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

5.2.1.7.88 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
```

Rückgabewert

Name	Typ
SetAttributeAsUlint	SXmlAttribute

Eingänge

Name	Typ
a	SXmlAttribute
value	ULINT

Beispielaufruf:

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

5.2.1.7.89 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
```

Rückgabewert

Name	Typ
SetChild	SXmlNode

Eingänge

Name	Typ
n	SXmlNode
cdata	BOOL

Ein-/Ausgänge

Name	Typ
value	STRING

Beispielaufruf:

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

5.2.1.7.90 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
```

 Rückgabewert

Name	Typ
SetChildAsBool	SXmlNode

 Eingänge

Name	Typ
n	SXmlNode
value	BOOL

Beispielaufruf:

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

5.2.1.7.91 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
```

 Rückgabewert

Name	Typ
SetChildAsDouble	SXmlNode

 Eingänge

Name	Typ
n	SXmlNode
value	LREAL

Beispielaufruf:

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

5.2.1.7.92 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
```

🔏 Rückgabewert

Name	Typ
SetChildAsFloat	SXmlNode

🔏 Eingänge

Name	Typ
n	SXmlNode
value	REAL

Beispielaufruf:

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

5.2.1.7.93 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
```

🔏 Rückgabewert

Name	Typ
SetChildAsInt	SXmlNode

🔏 Eingänge

Name	Typ
n	SXmlNode
value	DINT

Beispielaufruf:

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

5.2.1.7.94 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
```

 **Rückgabewert**

Name	Typ
SetChildAsLint	SXmlNode

 **Eingänge**

Name	Typ
n	SXmlNode
value	LINT

Beispielaufruf:

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

5.2.1.7.95 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
```

 **Rückgabewert**

Name	Typ
SetChildAsUint	SXmlNode

 **Eingänge**

Name	Typ
n	SXmlNode
value	UDINT

Beispielaufruf:

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

5.2.1.7.96 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
```

🔑 Rückgabewert

Name	Typ
SetChildAsUlint	SXmlNode

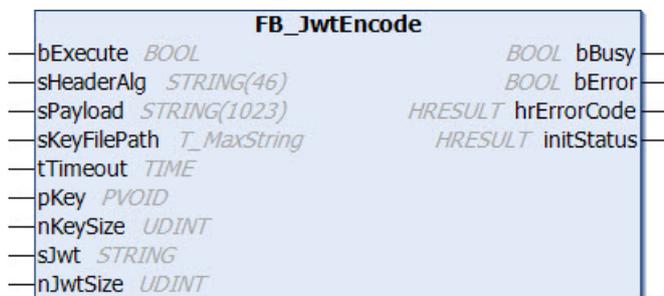
🔑 Eingänge

Name	Typ
n	SXmlNode
value	ULINT

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 <code>bError</code> 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
```

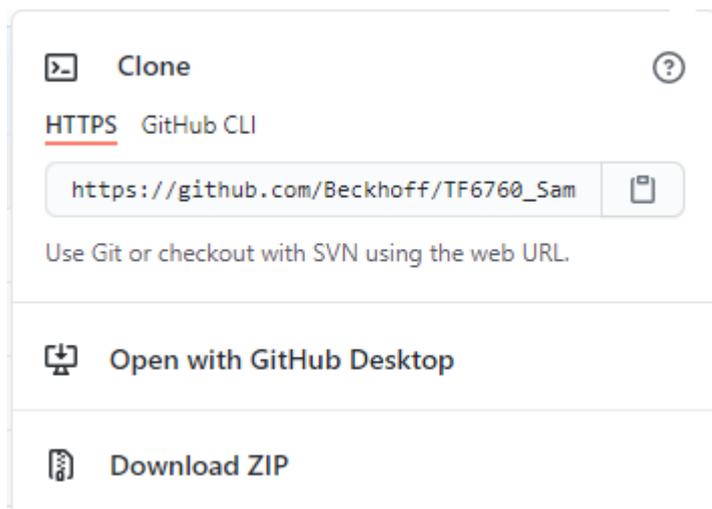
6 Beispiele

6.1 IoTHttpSamples

Jedes Beispiel wird als separates SPS-Projekt bereitgestellt. Die folgende Tabelle zeigt eine Auflistung der verfügbaren Beispiele:

Beispiel	Kurzbeschreibung
PostmanEcho	Postman Echo ist ein Test-Webservice für REST-Clients. Es sind Beispiele für HTTP GET, HTTP POST und HTTP PUT vorhanden. Außerdem gibt es ein weiteres HTTP GET-Beispiel mit Header-Authentifizierung.
OpenWeatherMap	OpenWeatherMap ist eine REST-API zur Abfrage von Wetterdaten. Im Beispiel werden Wetterdaten per HTTP GET im JSON-Format abgefragt.
Philips Hue	Philips Hue ist eine Smart Home-Lösung mit einer REST API. Per HTTP PUT werden im Beispiel Werte für eine Hue-Lampe verändert.
Telegram	Telegram ist ein Messenger mit einer REST API. Im Beispiel werden über einen sogenannten Telegram-Bot Nachrichten aus TwinCAT an einen Telegram-Benutzer gesendet.
AWS IoT	AWS IoT ist der Cloud-Dienst von Amazon Web Services. Im Beispiel werden ein HTTP GET an einen AWS IoT Shadow und ein HTTP POST an den AWS IoT Core Message Broker gezeigt.
AWS Service Configuration	Es ist über HTTP auch möglich, auf verschiedene AWS-Services zuzugreifen. In diesem Beispiel wird per HTTP GET auf einen AWS Service zugegriffen.

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.

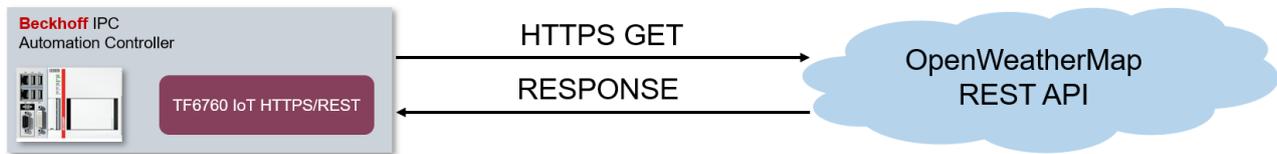


Die SPS-Projekte sind in einen MAIN-Teil und verschiedene Funktionsbausteine aufgeteilt. Die HTTP-befehlsspezifischen Funktionen (GET, POST, PUT) sind in den genannten Funktionsbausteinen gekapselt, um das Programm übersichtlich zu halten.

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>.

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.

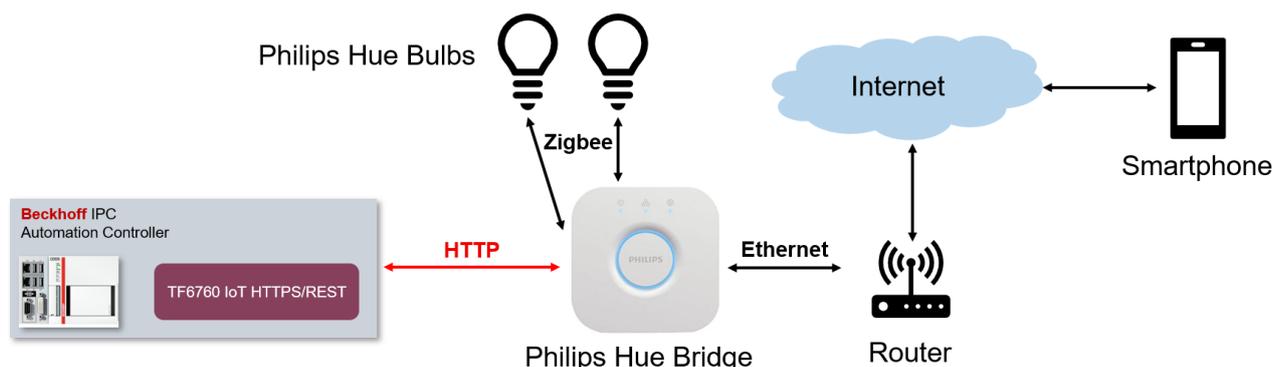
6.1.3 AWS IoT

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. Der Benutzer muss ein eigenes AWS-Konto haben, damit ein IoT Core-Endpoint vorhanden ist und die Möglichkeit besteht, Zertifikate für die Kommunikation mit TwinCAT zu erstellen.

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.



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.

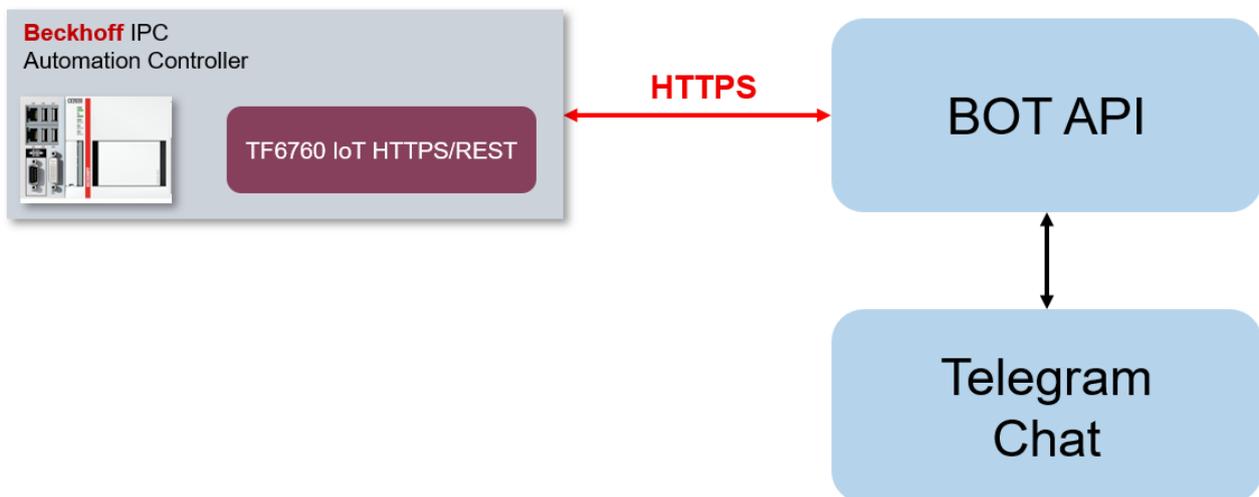
Philips bietet hier ein Tutorial „Erste Schritte“, in dem erläutert wird, wie die Bridge in einem Netzwerk zu finden ist und wie eine Client-Anwendung implementiert wird: <https://developers.meethue.com/develop/get-started-2/>

Die API-Beschreibung ist auf derselben Website zu finden, hierfür ist eine Registrierung erforderlich.

Das vorliegende Beispiel implementiert zum einen eine einfache PUT-Anfrage und zum anderen eine togglende 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).

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.



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-Father kontaktieren kann. Der BotFather ist ein allgemein verfügbarer Account für die Erstellung neuer Bots.

**Michael**

2:29:06 PM

/newbot

**BotFather**

2:29:06 PM

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

**Michael**

2:29:14 PM

BeckhoffWebinar

**BotFather**

2:29:14 PM

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

**Michael**

2:30:38 PM

BeckhoffIoTBot

**BotFather**

2:30:38 PM

Done! Congratulations on your new bot. You will find it at t.me/BeckhoffIoTBot. You can now add a description, about section and profile picture for your bot, see [/help](#) for a list of commands. By the way, when you've finished creating your cool bot, ping our Bot Support if you want a better username for it. Just make sure the bot is fully operational before you do this.

Use this token to access the HTTP API:

`1106704362:AAExFeda7Jf9CojjI9`

Keep your token **secure** and **store it safely**, it can be used by anyone to control your bot.

For a description of the Bot API, see this page:

<https://core.telegram.org/bots/api>



Write a message...

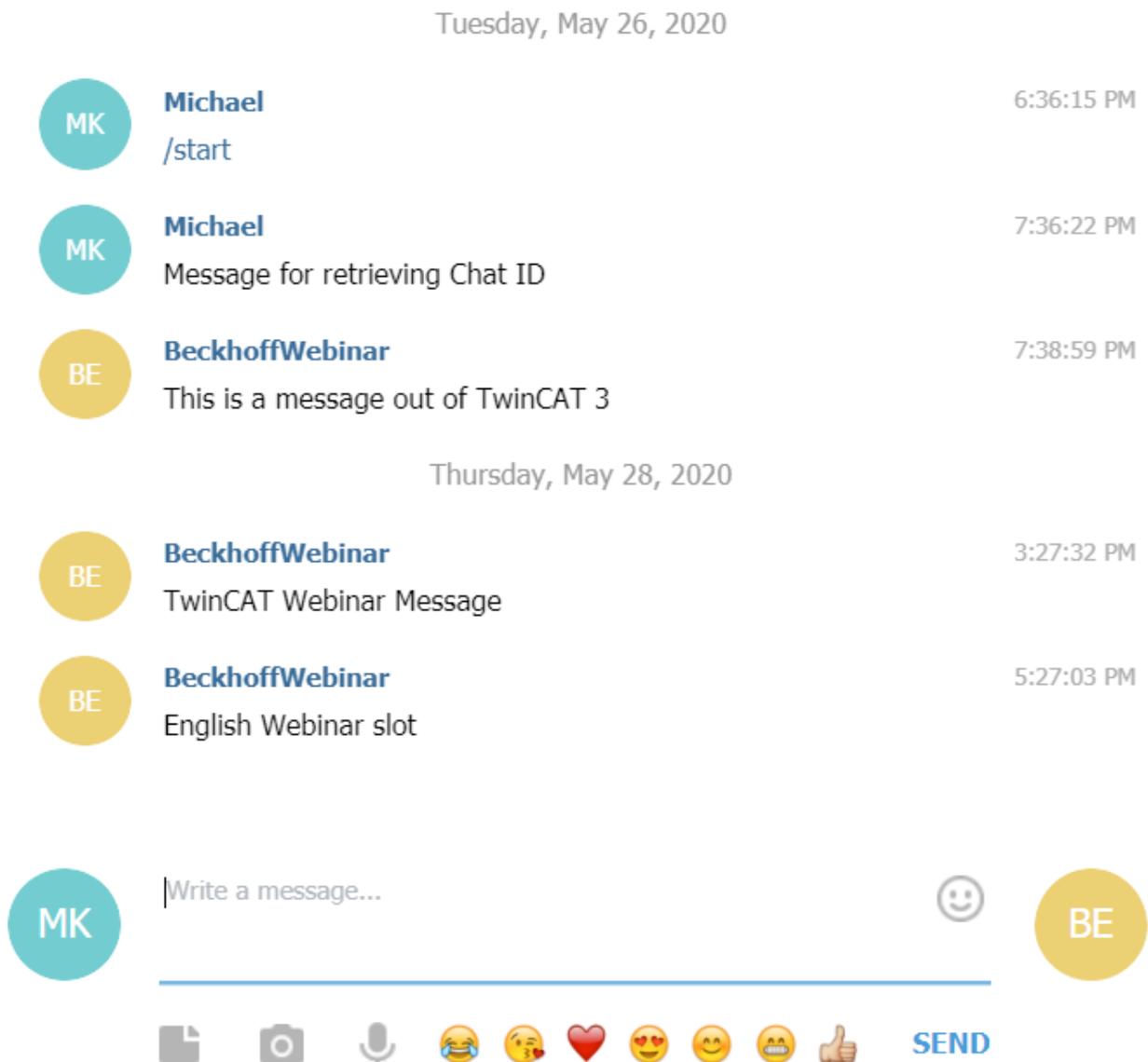


Der BotFather liefert ein API-Token zurück, das beim Senden von Anfragen aus TwinCAT an die Telegram-API verwendet werden muss. Mit Zugriff auf dieses Token kann ein HTTP-Client jede Funktion steuern, die der Bot bietet. Eine Beschreibung der BOT-API findet sich hier: <https://core.telegram.org/bots/api>

Grundsätzlich sind die Anfragen nach folgendem Schema aufgebaut:

`https://api.telegram.org/bot<token>/METHOD_NAME`

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. Zu diesem Zweck wird zunächst eine Nachricht über den Browser oder die Telegram-App in den Chat des Bots geschrieben.



Anschließend kann über die `getUpdates`-Methode die ChatID herausgefunden werden. Hier wird beispielhaft Postman als HTTP-Client benutzt:

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** `https://api.telegram.org/bot6792321317:AAHGRHC3eoaFCjrWGj08I0k67cju_bZnCXg/getUpdates`
- Params:** Authorization, Headers (7), Body, Pre-request Script, Tests, Settings
- Query Params:** A table with columns KEY, VALUE, and DESCRIPTION. The first row shows 'Key' and 'Value'.
- Status:** 200 OK, Time: 18ms, Size: 692 B
- Response Format:** Pretty, Raw, Preview, Visualize BETA, JSON
- Response Body (JSON):**

```

1  {
2    "ok": true,
3    "result": [
4      {
5        "update_id": [REDACTED],
6        "message": {
7          "message_id": 6,
8          "from": {
9            "id": [REDACTED],
10           "is_bot": false,
11           "first_name": "Michael",
12           "username": "[REDACTED]",
13           "language_code": "de"
14         },
15         "chat": {
16           "id": 5756769038,
17           "first_name": "Michael",

```

Unter Verwendung dieser Chat-ID kann dann die im Beispiel verwendete `sendMessage`-Methode zum Senden von Chat-Nachrichten aus dem TwinCAT-Programm verwendet werden.

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 \[► 24\]](#) reißt kurz den Hintergrund der AWS-eigenen Signierungsfunktion an. Alles andere entnehmen Sie bitte der AWS-Dokumentation.

Wie bereits im Kapitel [URL-Redirects \[► 24\]](#) 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.

● **Alphabetische Sortierung notwendig**

i 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 Benutzer selbst darauf achten, ansonsten kommt eine Fehlermeldung vom HTTP-Server.

7 Anhang

7.1 ADS Return Codes

Gruppierung der Fehlercodes:

Globale Fehlercodes: 0x0000 [▶ 245]... (0x9811_0000 ...)

Router Fehlercodes: 0x0500 [▶ 245]... (0x9811_0500 ...)

Allgemeine ADS Fehler: 0x0700 [▶ 246]... (0x9811_0700 ...)

RTime Fehlercodes: 0x1000 [▶ 248]... (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, nicht erreichbar oder nicht installiert.
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_INVALIDAMSLLENGTH	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_INVALIDAMSFAGMENT	Ungültiges AMS Fragment.
0x1D	29	0x9811001D	ERR_TLSSSEND	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. Mehrere Ursachen sind möglich. Beispielsweise beim Anlegen von Routen, dass ein falsches Passwort angegeben wurde.
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-Synchronisation in der SPS geben.
0x70A	1802	0x9811070A	ADSERR_DEVICE_NOMEMORY	Nicht genügend Speicher.
0x70B	1803	0x9811070B	ADSERR_DEVICE_INVALIDPARG	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_NOTIFYHANDINVALID	Notification Handle ist ungültig.
0x715	1813	0x98110715	ADSERR_DEVICE_CLIENTUNKNOWN	Notification-Client nicht registriert.
0x716	1814	0x98110716	ADSERR_DEVICE_NOMOREHDL	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. Mehrere Ursachen sind möglich. Beispielsweise, dass eine Unidirectionale ADS Route in die umgekehrte Richtung verwendet wird.
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.

Hex	Dec	HRESULT	Name	Beschreibung
0x732	1842	0x98110732	ADSERR_DEVICE_LICENSEDEMOTENIED	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.
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_INVALIDPARAM	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_SYNC_TIMEOUT	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_NOMORESVM	Keine Symbole im Cache.
0x754	1876	0x98110754	ADSERR_CLIENT_SYNCRESINVALID	Ungültige Antwort erhalten.
0x755	1877	0x98110755	ADSERR_CLIENT_SYNCPORTLOCKED	Sync Port ist verriegelt.
0x756	1878	0x98110756	ADSERR_CLIENT_REQUESTCANCELLED	Die Anfrage wurde abgebrochen.

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_IRQNOTLESSOREQUAL	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_VMXCONTROLSSMISSING	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			

Trademark statements

Beckhoff®, ATRO®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, MX-System®, Safety over EtherCAT®, TC/BSD®, TwinCAT®, TwinCAT/BSD®, TwinSAFE®, XFC®, XPlanar® and XTS® are registered and licensed trademarks of Beckhoff Automation GmbH.

Third-party trademark statements

Arm, Arm9 and Cortex are trademarks or registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere.

Intel, the Intel logo, Intel Core, Xeon, Intel Atom, Celeron and Pentium are trademarks of Intel Corporation or its subsidiaries.

Microsoft, Microsoft Azure, Microsoft Edge, PowerShell, Visual Studio, Windows and Xbox are trademarks of the Microsoft group of companies.

Mehr Informationen:
www.beckhoff.com/tf6760/

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Deutschland
Telefon: +49 5246 9630
info@beckhoff.com
www.beckhoff.com

