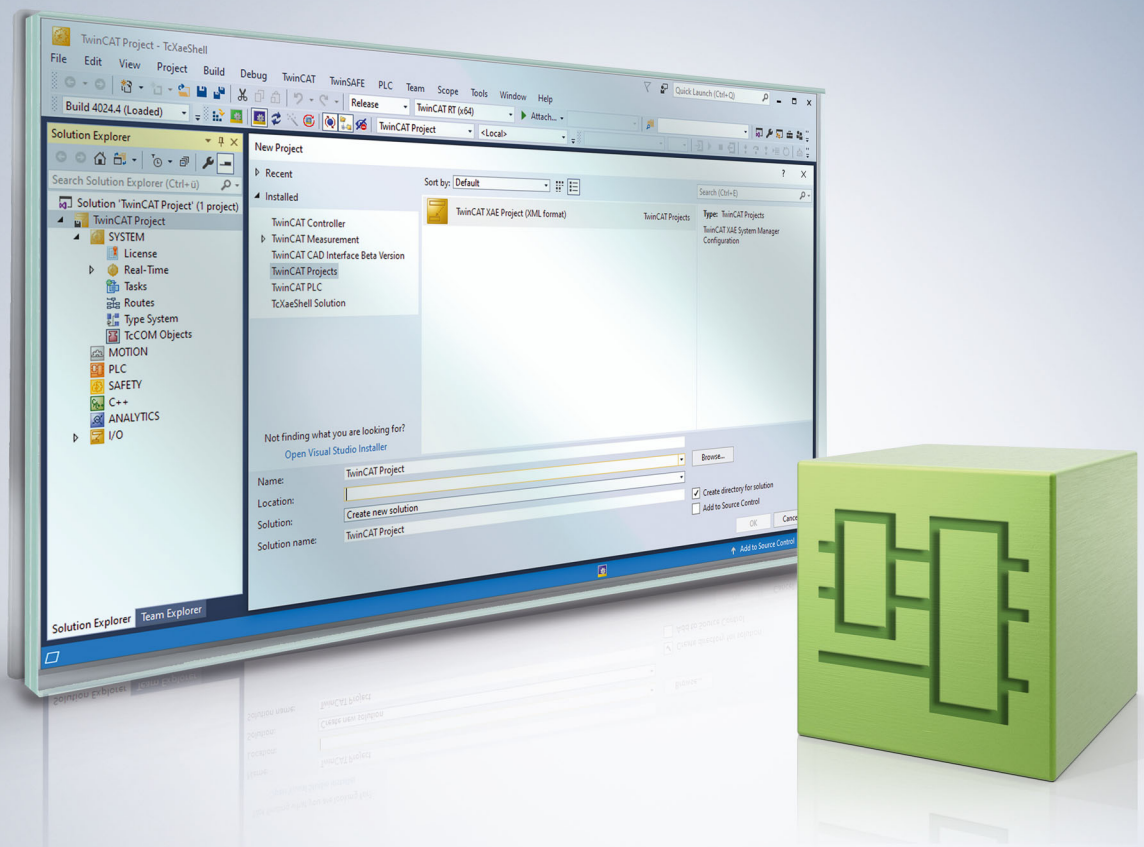
BECKHOFF New Automation Technology

Handbuch | DE

TE1000

TwinCAT 3 | PLC Lib: Tc3_JsonXml



15.03.2024 | Version: 1.13.0

Inhaltsverzeichnis

1 Vorwort

1.1 Hinweise zur Dokumentation

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs- und Automatisierungstechnik, das mit den geltenden nationalen Normen vertraut ist.
Zur Installation und Inbetriebnahme der Komponenten ist die Beachtung der Dokumentation und der nachfolgenden Hinweise und Erklärungen unbedingt notwendig.
Das Fachpersonal ist verpflichtet, 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®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® und XPlanar® sind eingetragene und lizenzierte Marken der Beckhoff Automation GmbH.
Die Verwendung anderer in dieser Dokumentation enthaltenen Marken oder Kennzeichen durch Dritte kann zu einer Verletzung von Rechten der Inhaber der entsprechenden Bezeichnungen führen.

Patente

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

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

Copyright

1.2 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.

1.4 Ausgabestände der Dokumentation

Version	Änderung
1.13.0	**Neu:** Funktionsbausteine/FB_JsonDomParser/PushbackCopyValue und /FB_XmlDomParser/ClearIterator

2 Übersicht

Mithilfe der SPS-Bibliothek Tc3_JsonXml können SAX- und DOM-Parser-Technologien zur Erstellung und zum Navigieren von JSON- und XML-Dokumenten verwendet werden.

Systemvoraussetzung

Target System	Win7, WES7, WEC7, Win10 IPC or CX, (x86, x64, ARM)
Min. TwinCAT-Version	3.1.4022.0
Min. TwinCAT-Level	TC1200 TC3 PLC

SAX (Simple API for XML)

SAX wurde ursprünglich zur Behandlung von XML-Dokumenten entwickelt, kann aber auch für andere Datenformate, wie z. B. JSON, verwendet werden. Ein SAX-Parser behandelt die zu lesenden oder schreibenden Daten als sequentiellen Datenstrom. Beim Lesen eines Datenstroms werden definierte Callback-Methoden aufgerufen, welche dann die entsprechenden Inhalte des Datenstroms zurückliefern. Bei einem SAX-Parser wird daher auch von einem Event-basierten Parser gesprochen. Die auftretenden Ereignisse (Callback-Methoden) sind zustandslos, d. h. sie hängen nicht von vorhergehenden Ereignissen ab. Der Vorteil hierbei ist, dass das XML-Dokument zu keinem Zeitpunkt komplett im Speicher gehalten werden muss und die Anwendung über die Callbacks „on the fly“ reagieren kann.

DOM (Document Object Model)

DOM ist eine Spezifikation für den Zugriff auf XML-Dokumente, kann aber auch für andere Datenformate, wie z. B. HTML oder JSON, verwendet werden. Der Schnittstelle liegt ein definiertes Objektmodell zugrunde, dessen Gültigkeit eine Voraussetzung für die korrekte Verwendung ist. Dieses Objektmodell repräsentiert ein Dokument, z. B. ein JSON-Dokument, in Form einer Baumstruktur im Speicher, welche dann zur Navigation durch das Dokument verwendet werden kann. DOM erlaubt hierbei die Navigation zwischen den einzelnen Knoten, das Erzeugen, Verschieben und Löschen von Knoten, sowie das Auslesen, Ändern und Löschen von Knoteninhalten. Am Ende der Verarbeitung wird aus der finalisierten Baumstruktur dann ein neues JSON- oder XML-Dokument generiert. Der Vorteil hierbei ist, dass kein eigener Datenhaushalt mit den eingelesenen Daten erstellt werden muss, da die Daten im DOM vorliegen und immer wieder auf sie zugegriffen werden kann.

JSON-Dokument

Der folgende Ausschnitt zeigt ein beispielhaftes JSON-Dokument:

```
{
  "VariableNameX": 0.0,
  "VariableNameY": 0.0,
  "VariableNameZ": 0.0
}
```

Metadaten

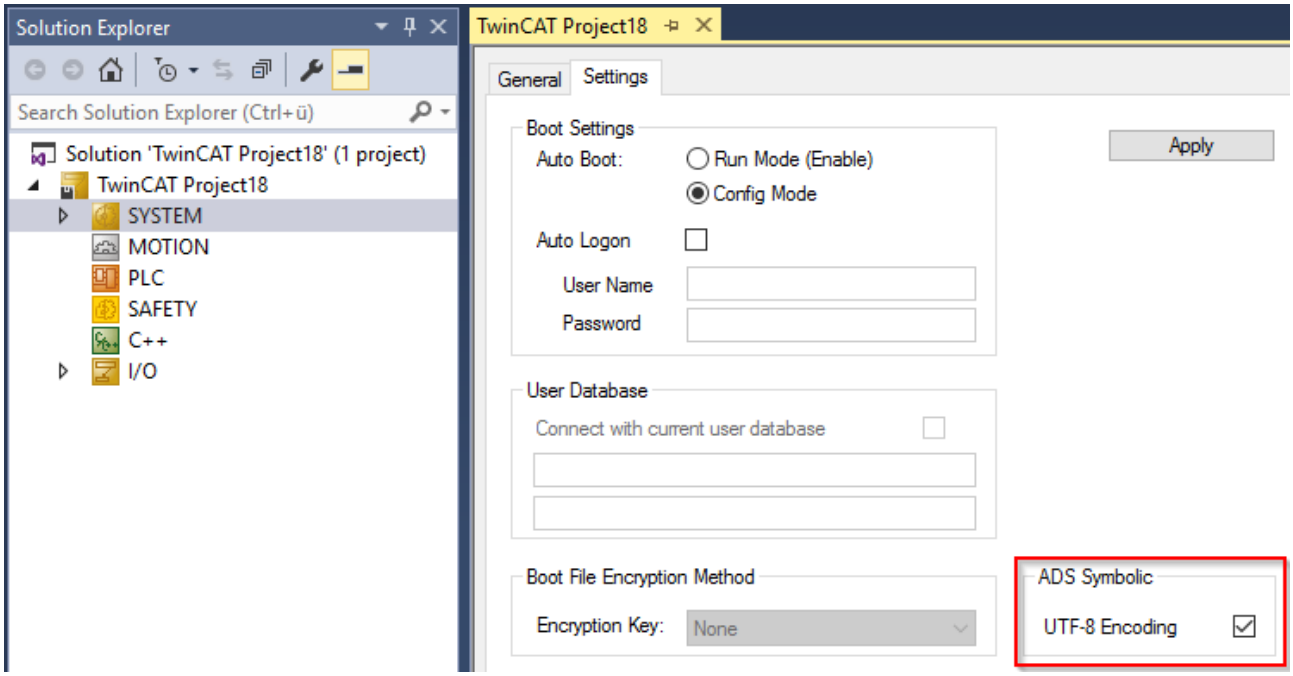
Die Tc3_JsonXml-Bibliothek beinhaltet den Funktionsbaustein FB_JsonReadWriteDataType [▶ 133], der eine automatische Generierung von Metadaten anhand von SPS-Attributen ermöglicht.

```
{
  "Values": {
    "VariableNameX": 0.0,
    "VariableNameY": 0.0,
    "VariableNameZ": 0.0
  },
  "MetaData": {
    "VariableNameX": {
      "Unit": "A"
    },
    "VariableNameY": {
      "Unit": "V"
    },
    "VariableNameZ": {
```

```
      "Unit": "mA"
    }
  }
}
```

Siehe auch: Beispiele > Tc3JsonXmlSampleJsonDataType [▶ 219].

Zur Verwendung von UTF-8-Zeichen, z. B. bei der automatischen Generierung von Metadaten über den Funktionsbaustein FB_JsonReadWriteDataType [▶ 133], 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.



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 Tc3_JsonXml Bibliothek stellt über die Methode FB_JwtEncode [▶ 212] eine Funktion zum Erzeugen eines JWT zur Verfügung.

3 Getting Started

Dieser Dokumentationsartikel soll Ihnen einen ersten, schnellen Start in die Verwendung der Tc3_JsonXml SPS-Bibliothek ermöglichen. Der Artikel geht hierbei von folgendem Anwendungsfall aus:

- Es existiert ein JSON Dokument, welches weiterverarbeitet werden soll.
- Das JSON Dokument liegt in Form einer Datei auf dem lokalen Dateisystem.
- Der Artikel beschreibt die einzelnen Schritte zum Parsen des JSON Dokuments.

Für weitere Code-Beispiele empfehlen wir auch unsere Samples [▶ 219].

Inhalt der Datei

Die Datei *myJsonContent.json* liegt im lokalen Dateisystem unter folgendem Pfad: *c:\temp\myJsonContent.json* und hat den folgenden Inhalt:

```
{
  "Timestamp":"2017-04-04T12:42:42",
  "Values": {
      "Sensor1": 42.41999816894531,
      "Sensor2": [1,2,3,4,5],
      "Sensor3": 10
  }
}
```

Deklarationen

Die folgenden Variablen-Deklarationen werden für das weitere Vorgehen benötigt:

```
fbJson : FB_JsonDomParser;
jsonRoot : SJsonValue;
jsonProp : SJsonValue;
jsonIterator : SJsonValue;
jsonIteratorEnd : SJsonValue;
bLoadJsonFile : BOOL;
sTimestamp : STRING;
fSensor1 : LREAL;
aSensor2 : ARRAY[0..4] OF DINT;
nSensor3 : DINT;
i : UINT;
```

Bitte stellen Sie des Weiteren sicher, dass Sie eine Referenz zu der SPS-Bibliothek Tc3_JsonXml zu Ihrem SPS-Projekt hinzugefügt haben.

Einlesen der Datei

Zum Einlesen der Datei wird die Methode LoadDocumentFromFile [▶ 62] aus dem Funktionsbaustein FB_JsonDomParser [▶ 19] verwendet. Die Ausführung der Methode wird über eine steigende Flanke am Eingang `bExec` gesteuert.

```
IF bLoadJsonFile = TRUE THEN
  fbJson.LoadDocumentFromFile('C:\Temp\myJsonContent.json', bLoadJsonFile);
END_IF
```

Quelle des JSON Dokuments

Falls in Ihrem Anwendungsfall das JSON Dokument bereits in der SPS vorliegt, z. B. in einer Variablen vom Datentyp STRING, so können Sie die Methode ParseDocument [▶ 65] verwenden, um das Dokument in den Speicher zu laden und weiter zu verarbeiten.

Parsen des JSON Dokuments

Mit der Methode GetDocumentRoot [▶ 47] lässt sich der Anfang des JSON Dokuments im Speicher referenzieren. Der Rückgabewert der Methode ist ein Interface Pointer.

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

Von hier können nun die weiteren Keys auf erster Ebene ausgelesen werden, z. B. der Key 'Timestamp' über die Methode GetString [▶ 52]:

```
jsonProp := fbJson.FindMember(jsonRoot, 'Timestamp');
IF (jsonProp <> 0) THEN
  sTimestamp := fbJson.GetString(jsonProp);
END_IF
```

Bei dem nächsten Key ('Values') handelt es sich um ein verschachteltes JSON Objekt. Die Kindelemente von diesem Objekt können über die Methode FindMemberPath [▶ 41] direkt ausgelesen werden, zum Beispiel für das Element 'Values/Sensor1':

```
jsonProp := fbJson.FindMemberPath(jsonRoot, 'Values/Sensor1');
IF (jsonProp <> 0) THEN
  fSensor1 := fbJson.GetDouble(jsonProp);
END_IF
```

Bei dem nächsten Kindelement ('Values/Sensor2') handelt es sich um ein Array. Dieses kann über die Verwendung der Methoden ArrayBegin [▶ 35] und ArrayEnd [▶ 35] ausgelesen werden.

```
jsonProp := fbJson.FindMemberPath(jsonRoot, 'Values/Sensor2');
IF (jsonProp <> 0) THEN
  jsonIterator := fbJson.ArrayBegin(jsonProp);
  jsonIteratorEnd := fbJson.ArrayEnd(jsonProp);
  WHILE jsonIterator <> jsonIteratorEnd DO
    IF (jsonProp <> 0) THEN
      aSensor2[i] := fbJson.GetInt(jsonIterator);
    END_IF
    jsonIterator := fbJson.NextArray(jsonIterator);
    i := i + 1;
  END_WHILE
  i := 0;
END_IF
```

Das nächste Kindelement ('Values/Sensor3') ist ähnlich in der Handhabung wie das Element 'Values/Sensor1'. Anstelle der Methode GetDouble [▶ 47] wird hier dann jedoch die Methode GetInt [▶ 48] verwendet, um den Wert des Keys auszulesen.

```
jsonProp := fbJson.FindMemberPath(jsonRoot, 'Values/Sensor3');
IF (jsonProp <> 0) THEN
  nSensor3 := fbJson.GetInt(jsonProp);
END_IF
```

Komplettes Codebeispiel

Im Folgenden finden Sie die obigen Code-Snippets einmal als vollständiges Beispiel.

Deklarationsteil

```
PROGRAM MAIN
  fbJson : FB_JsonDomParser;
  jsonRoot : SJsonValue;
  jsonProp : SJsonValue;
  jsonIterator : SJsonValue;
  jsonIteratorEnd : SJsonValue;
  bLoadJsonFile : BOOL;
  sTimestamp : STRING;
  fSensor1 : LREAL;
  aSensor2 : ARRAY[0..4] OF DINT;
  nSensor3 : DINT;
  i : UINT;
END_VAR
```

Implementierungsteil

```
IF bLoadJsonFile = TRUE THEN
  fbJson.LoadDocumentFromFile('C:\Temp\myJsonContent.json', bLoadJsonFile);
  jsonRoot := fbJson.GetDocumentRoot();
  jsonProp := fbJson.FindMember(jsonRoot, 'Timestamp');
  IF (jsonProp <> 0) THEN
    sTimestamp := fbJson.GetString(jsonProp);
  END_IF
  jsonProp := fbJson.FindMemberPath(jsonRoot, 'Values/Sensor1');
  IF (jsonProp <> 0) THEN
    fSensor1 := fbJson.GetDouble(jsonProp);
  END_IF
  jsonProp := fbJson.FindMemberPath(jsonRoot, 'Values/Sensor2');
  IF (jsonProp <> 0) THEN
    jsonIterator := fbJson.ArrayBegin(jsonProp);
    jsonIteratorEnd := fbJson.ArrayEnd(jsonProp);
    WHILE jsonIterator <> jsonIteratorEnd DO
```

```
      IF (jsonProp <> 0) THEN
        aSensor2[i] := fbJson.GetInt(jsonIterator);
      END_IF
      jsonIterator := fbJson.NextArray(jsonIterator);
      i := i + 1;
    END_WHILE
    i := 0;
  END_IF
  jsonProp := fbJson.FindMemberPath(jsonRoot, 'Values/Sensor3');
  IF (jsonProp <> 0) THEN
    nSensor3 := fbJson.GetInt(jsonProp);
  END_IF
END_IF
```

Nach dem erfolgreichen Durchlauf des obigen Codes befinden sich die ausgelesenen Keys in den entsprechenden SPS Variablen, z. B. wie folgt:

MAIN [Online]

TwinCAT_Project1.Untitled1.MAIN

Expression	Type	Value
fbJson	FB_JsonDomParser	
jsonDoc	SJsonValue	0000000000000...
jsonRoot	SJsonValue	ffffb209f5807908
jsonProp	SJsonValue	ffffb209f58083f8
jsonValue	SJsonValue	0000000000000...
jsonIterator	SJsonValue	ffffb209f5808380
jsonIteratorEnd	SJsonValue	ffffb209f5808380
bHasMember	BOOL	FALSE
bLoadJsonFile	BOOL	FALSE
sTimestamp	STRING	'2017-04-04T12:...
lrSensor1	LREAL	42.41999816894...
arrSensor2	ARRAY [0..4] OF DINT	
arrSensor2[0]	DINT	1
arrSensor2[1]	DINT	2
arrSensor2[2]	DINT	3
arrSensor2[3]	DINT	4
arrSensor2[4]	DINT	5
nSensor3	DINT	10
i	UINT	0

4 Funktionsbausteine

4.1 FB_JsonDomParser

Dieser Funktionsblock ist von demselben internen Funktionsbaustein abgeleitet wie der FB_JsonDynDomParser [▶ 88] 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 [▶ 88] 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 [▶ 64]() oder ParseDocument [▶ 65]() 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 [▶ 24]	Fügt ein Array-Member zu einem JSON-Objekt hinzu.
AddBase64Member [▶ 24]	Fügt ein Base64-Member zu einem JSON-Objekt hinzu.
AddBoolMember [▶ 25]	Fügt ein Bool-Member zu einem JSON-Objekt hinzu.
AddDateTimeMember [▶ 26]	Fügt ein DateTime-Member zu einem JSON-Objekt hinzu.
AddDcTimeMember [▶ 27]	Fügt ein DcTime-Member zu einem JSON-Objekt hinzu.
AddDoubleMember [▶ 27]	Fügt ein Double-Member zu einem JSON-Objekt hinzu.
AddFileTimeMember [▶ 28]	Fügt ein FileTime-Member zu einem JSON-Objekt hinzu.
AddHexBinaryMember [▶ 29]	Fügt ein HexBinary-Member zu einem JSON-Objekt hinzu.
AddInt64Member [▶ 29]	Fügt ein Int64-Member zu einem JSON-Objekt hinzu.
AddIntMember [▶ 30]	Fügt ein Int-Member zu einem JSON-Objekt hinzu.
AddJsonMember [▶ 31]	Fügt ein JSON-Member zu einem JSON-Objekt hinzu.
AddNullMember [▶ 31]	Fügt ein NULL-Member zu einem JSON-Objekt hinzu.
AddObjectMember [▶ 32]	Fügt ein Object-Member zu einem JSON-Objekt hinzu.
AddStringMember [▶ 33]	Fügt ein String-Member zu einem JSON-Objekt hinzu.
AddUint64Member [▶ 33]	Fügt ein UInt64-Member zu einem JSON-Objekt hinzu.
AddUintMember [▶ 34]	Fügt ein UInt-Member zu einem JSON-Objekt hinzu.
ArrayBegin [▶ 35]	Liefert das erste Element eines Arrays.
ArrayEnd [▶ 35]	Liefert das letzte Element eines Arrays.
ClearArray [▶ 36]	Löscht den Inhalt eines Arrays.
CopyDocument [▶ 37]	Kopiert den Inhalt des DOM-Speichers in eine Variable vom Datentyp STRING.
CopyJson [▶ 38]	Extrahiert ein JSON-Objekt aus einem Key und speichert dieses in einer Variablen vom Datentyp STRING.
CopyString [▶ 39]	Kopiert den Wert eines Keys in eine Variable vom Datentyp STRING.
FindMember [▶ 40]	Sucht in einem JSON-Dokument nach einem bestimmten Property.
FindMemberPath [▶ 41]	Sucht in einem JSON-Dokument nach einem bestimmten Property (Pfad-spezifisch).
GetArraySize [▶ 42]	Liefert die Anzahl der Elemente in einem JSON-Array.
GetArrayValue [▶ 43]	Liefert den Wert an der aktuellen Iterator-Position eines Arrays.
GetArrayValueByIdx [▶ 43]	Liefert den Wert eines Arrays an einem angegebenen Index.
GetBase64 [▶ 44]	Dekodiert einen Base64-Wert aus einem JSON-Property.
GetBool [▶ 44]	Liefert den Value eines Properties vom Datentyp BOOL.
GetDateTime [▶ 45]	Liefert den Value eines Properties vom Datentyp DATE_AND_TIME.
GetDcTime [▶ 45]	Liefert den Values eines Properties vom Datentyp DCTIME.
GetDocument [▶ 46]	Gibt den Inhalt des DOM-Speichers als Datentyp STRING(255) zurück.
GetDocumentLength [▶ 46]	Gibt die Länge eines JSON-Dokuments im DOM-Speicher zurück.
GetDocumentRoot [▶ 47]	Liefert den Root-Knoten eines JSON-Dokuments im DOM-Speicher.
GetDouble [▶ 47]	Liefert den Value eines Properties vom Datentyp LREAL.
GetFileTime [▶ 47]	Liefert den Value eines Properties vom Datentyp DCTIME.
GetHexBinary [▶ 48]	Dekodiert den HexBinary-Inhalt eines Properties und schreibt diesen an eine bestimmte Speicheradresse,
GetInt [▶ 48]	Liefert den Value eines Properties vom Datentyp DINT.
GetInt64 [▶ 49]	Liefert den Value eines Properties vom Datentyp LINT.
GetJson [▶ 49]	Liefert den Value eines Properties als Datentyp STRING(255) zurück.

Name	Beschreibung
GetJsonLength [▶ 50]	Liefert die Länge eines Properties, wenn dieses ein JSON-Dokument ist.
GetMaxDecimalPlaces [▶ 50]	Liefert die aktuelle Einstellung für MaxDecimalPlaces.
GetMemberName [▶ 51]	Liefert den Namen eines JSON-Property-Members an der Position des aktuellen Iterators,
GetMemberValue [▶ 51]	Liefert den Value eines JSON-Property-Members an der Position des aktuellen Iterators,
GetString [▶ 52]	Liefert den Value eines Properties vom Datentyp STRING(255).
GetStringLength [▶ 52]	Liefert die Länge eines Properties, wenn dessen Value ein String ist.
GetType [▶ 53]	Liefert den Typ eines Property-Values.
GetUint [▶ 54]	Liefert den Value eines Properties vom Datentyp UDINT.
GetUint64 [▶ 54]	Liefert den Value eines Properties vom Datentyp ULINT.
HasMember [▶ 54]	Prüft, ob ein bestimmtes Property im DOM-Speicher vorhanden ist.
IsArray [▶ 55]	Prüft, ob es sich bei einem gegebenen Property um ein Array handelt.
IsBase64 [▶ 56]	Prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp Base64 handelt.
IsBool [▶ 56]	Prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp BOOL handelt.
IsDouble [▶ 56]	Prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp Double (SPS: LREAL) handelt.
IsFalse [▶ 57]	Prüft, ob der Value eines gegebenen Properties FALSE ist.
IsHexBinary [▶ 57]	Prüft, ob der Value eines Properties ein HexBinary-Format hat.
IsInt [▶ 58]	Prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp Integer (SPS: DINT) handelt.
IsInt64 [▶ 58]	Prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp LINT handelt.
IsISO8601TimeFormat [▶ 59]	Prüft, ob es sich bei dem Value eines gegebenen Properties um ein Zeitformat laut ISO8601 handelt.
IsNull [▶ 59]	Prüft, ob es sich bei dem Value eines gegebenen Properties um NULL handelt.
IsNumber [▶ 60]	Prüft, ob es sich bei dem Value eines gegebenen Properties um einen numerischen Wert handelt.
IsObject [▶ 60]	Prüft, ob es sich bei dem gegebenen Property um ein weiteres JSON-Objekt handelt.
IsString [▶ 61]	Prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp STRING handelt.
IsTrue [▶ 61]	Prüft, ob der Wert eines gegebenen Properties TRUE ist.
IsUint [▶ 61]	Prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp UDINT handelt.
IsUint64 [▶ 62]	Prüft, ob es sich bei dem Value eines gegebenen Properties um den Datentyp ULINT handelt.
LoadDocumentFromFile [▶ 62]	Lädt ein JSON-Dokument aus einer Datei.
MemberBegin [▶ 63]	Liefert das erste Kindelement unterhalb eines JSON-Properties.
MemberEnd [▶ 64]	Liefert das letzte Kindelement unterhalb eines JSON-Properties.
NewDocument [▶ 64]	Erzeugt ein neues, leeres JSON-Dokument im DOM-Speicher.
NextArray [▶ 65]	Liefert das nächste Element in einem Array.
NextMember	Liefert das nächste Property in einem JSON-Dokument.
ParseDocument [▶ 65]	Lädt ein JSON-Objekt zur weiteren Verarbeitung in den DOM-Speicher.
PopbackValue	Löscht das Element am Ende eines Arrays.

Name	Beschreibung
PushbackBase64Value [▶ 66]	Hängt einen Base64-Wert an das Ende eines Arrays an.
PushbackBoolValue [▶ 66]	Hängt einen Base64-Wert an das Ende eines Arrays an.
PushbackDateTimeValue [▶ 67]	Hängt einen Wert vom Datentyp DATE_AND_TIME an das Ende eines Arrays an.
PushbackDcTimeValue [▶ 68]	Hängt einen Wert vom Datentyp DCTIME an das Ende eines Arrays an.
PushbackDoubleValue [▶ 69]	Hängt einen Wert vom Datentyp Double an das Ende eines Arrays an.
PushbackFileTimeValue [▶ 69]	Hängt einen Wert vom Datentyp FILETIME an das Ende eines Arrays an.
PushbackHexBinaryValue [▶ 70]	Hängt einen HexBinary-kodierten Wert an das Ende eines Arrays an.
PushbackInt64Value [▶ 70]	Hängt einen Wert vom Datentyp Int64 an das Ende eines Arrays an.
PushbackIntValue [▶ 71]	Hängt einen Wert vom Datentyp INT an das Ende eines Arrays an.
PushbackJsonValue [▶ 71]	Fügt ein JSON-Dokument zum Ende eines Arrays hinzu.
PushbackNullValue [▶ 72]	Hängt einen NULL-Wert an das Ende eines Arrays an.
PushbackStringValue [▶ 73]	Hängt einen Wert vom Datentyp String an das Ende eines Arrays an.
PushbackUint64Value [▶ 73]	Hängt einen Wert vom Datentyp UInt64 an das Ende eines Arrays an.
PushbackUintValue [▶ 74]	Hängt einen Wert vom Datentyp UInt an das Ende eines Arrays an.
RemoveAllMembers [▶ 74]	Entfernt alle Kindelemente von einem gegebenen Property.
RemoveArray [▶ 75]	Löscht den Wert des aktuellen Array-Iterators.
RemoveMember [▶ 76]	Löscht das Property an dem aktuellen Iterator.
RemoveMemberByName [▶ 76]	Entfernt ein Kindelement von einem gegebenen Property.
SaveDocumentToFile [▶ 77]	Speichert ein JSON-Dokument in einer Datei.
SetArray [▶ 78]	Setzt den Value eines Properties auf den Typ „Array“.
SetBase64 [▶ 79]	Setzt den Value eines Properties auf einen Base64-kodierten Wert.
SetBool [▶ 80]	Setzt den Value eines Properties auf einen Wert vom Datentyp BOOL.
SetDateTime [▶ 80]	Setzt den Value eines Properties auf einen Wert vom Datentyp DATE_AND_TIME.
SetDcTime [▶ 81]	Setzt den Value eines Properties auf einen Wert vom Datentyp DCTIME.
SetDouble [▶ 81]	Setzt den Value eines Properties auf einen Wert vom Datentyp Double.
SetFileTime [▶ 82]	Setzt den Value eines Properties auf einen Wert vom Datentyp FILETIME.
SetHexBinary [▶ 82]	Setzt den Value eines Properties auf einen HexBinary-kodierten Wert.
SetInt [▶ 83]	Setzt den Value eines Properties auf einen Wert vom Datentyp INT
SetInt64 [▶ 83]	Setzt den Value eines Properties auf einen Wert vom Datentyp Int64.
SetJson [▶ 84]	Fügt in den Value eines Properties ein weiteres JSON-Dokument ein.
SetMaxDecimalPlaces [▶ 85]	Setzt die aktuelle Einstellung für MaxDecimalPlaces.
SetNull [▶ 85]	Setzt den Value eines Properties auf den Wert NULL.
SetObject [▶ 86]	Setzt den Value eines Properties auf den Typ „Object“.
SetString [▶ 86]	Setzt den Value eines Properties auf einen Wert vom Datentyp STRING.
SetUint [▶ 87]	Setzt den Value eines Properties auf einen Wert vom Datentyp UInt.
SetUint64 [▶ 87]	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

4.1.1 AddArrayMember

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

Syntax

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

Rückgabewert

Name	Typ
AddArrayMember	SJsonValue

Eingänge

Name	Typ
v	SJsonValue
reserve	UDINT

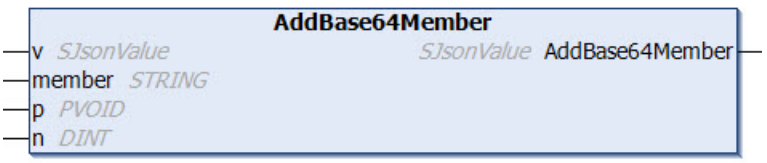
Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

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

4.1.2 AddBase64Member



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

Syntax

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

Rückgabewert

Name	Typ
AddBase64Member	SJsonValue

Eingänge

Name	Typ
v	SJsonValue
p	PVOID
n	DINT

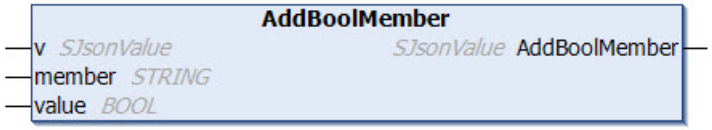
/ Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

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

4.1.3 AddBoolMember



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

Syntax

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

Rückgabewert

Name	Typ
AddBoolMember	SJsonValue

Eingänge

Name	Typ
v	SJsonValue
value	BOOL

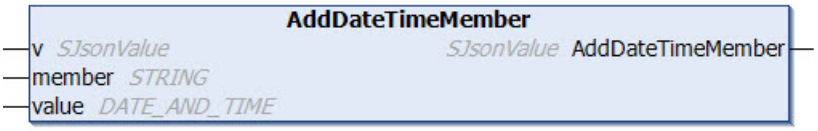
/ Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

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

4.1.4 AddDateTimeMember



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

Syntax

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

Rückgabewert

Name	Typ
AddDateTimeMember	SJsonValue

Eingänge

Name	Typ
v	SJsonValue
value	DATE_AND_TIME

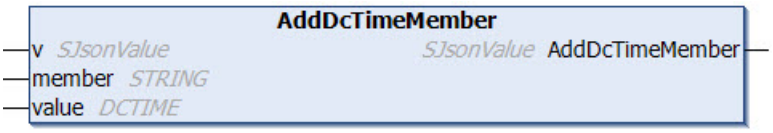
/ Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

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

4.1.5 AddDcTimeMember



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

Syntax

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

Rückgabewert

Name	Typ
AddDcTimeMember	SJsonValue

Eingänge

Name	Typ
v	SJsonValue
value	DCTIME

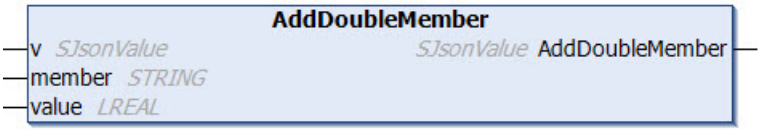
Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

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

4.1.6 AddDoubleMember



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

Syntax

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

Rückgabewert

Name	Typ
AddDoubleMember	SJsonValue

Eingänge

Name	Typ
v	SJsonValue
value	LREAL

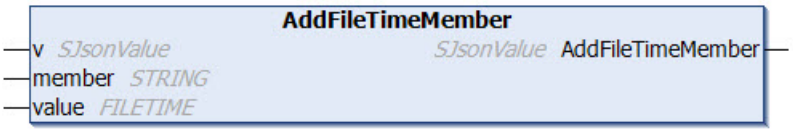
/ Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

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

4.1.7 AddFileTimeMember



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

Syntax

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

Rückgabewert

Name	Typ
AddFileTimeMember	SJsonValue

Eingänge

Name	Typ
v	SJsonValue
value	FILETIME

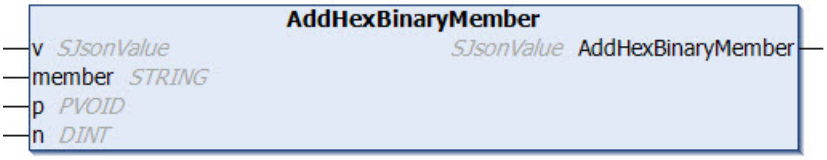
/ Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

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

4.1.8 AddHexBinaryMember



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

Syntax

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

Rückgabewert

Name	Typ
AddHexBinaryMember	SjsonValue

Eingänge

Name	Typ
v	SjsonValue
p	PVOID
n	DINT

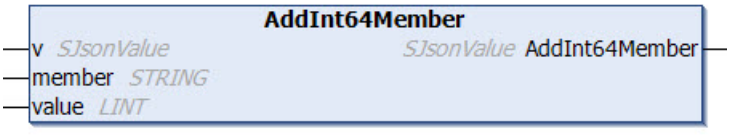
/ Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

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

4.1.9 AddInt64Member



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

Syntax

```
METHOD AddFileTimeMember : SjsonValue
VAR_INPUT
  v       : SjsonValue;
  value   : LINT;
```

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

Rückgabewert

Name	Typ
AddInt64Member	SJsonValue

Eingänge

Name	Typ
v	SJsonValue
value	LINT

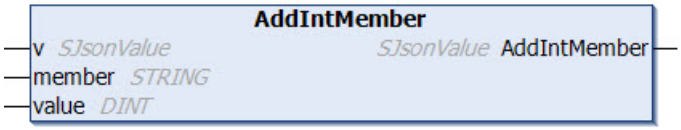
/ Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

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

4.1.10 AddIntMember



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

Syntax

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

Rückgabewert

Name	Typ
AddIntMember	SJsonValue

Eingänge

Name	Typ
v	SJsonValue
value	DINT

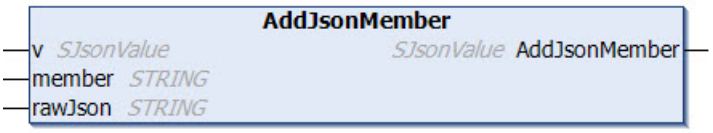
/ Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

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

4.1.11 AddJsonMember



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

Syntax

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

Rückgabewert

Name	Typ
AddJsonMember	SJsonValue

Eingänge

Name	Typ
v	SJsonValue

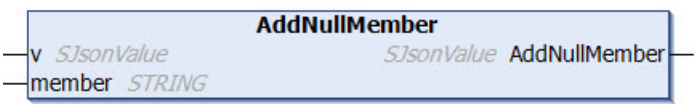
/ Ein-/Ausgänge

Name	Typ
member	STRING
rawJson	STRING

Beispielaufruf:

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

4.1.12 AddNullMember



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

Syntax

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

Rückgabewert

Name	Typ
AddNullMember	SJsonValue

Eingänge

Name	Typ
v	SJsonValue

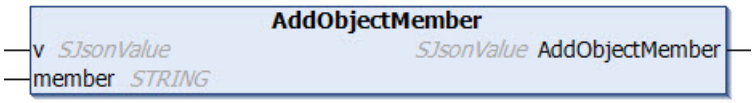
/ Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

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

4.1.13 AddObjectMember



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

Syntax

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

Rückgabewert

Name	Typ
AddObjectMember	SJsonValue

Eingänge

Name	Typ
v	SJsonValue

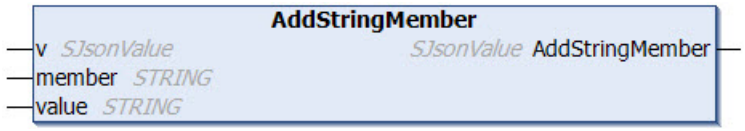
/ Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

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

4.1.14 AddStringMember



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

Syntax

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

Rückgabewert

Name	Typ
AddStringMember	SJsonValue

Eingänge

Name	Typ
v	SJsonValue

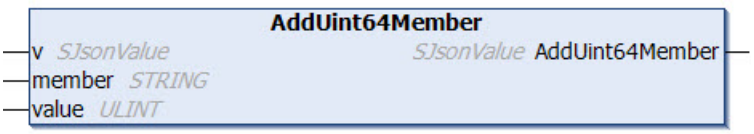
/ Ein-/Ausgänge

Name	Typ
member	STRING
value	STRING

Beispielaufruf:

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

4.1.15 AddUint64Member



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

Syntax

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

Rückgabewert

Name	Typ
AddUint64Member	SJsonValue

Eingänge

Name	Typ
v	SJsonValue
value	ULINT

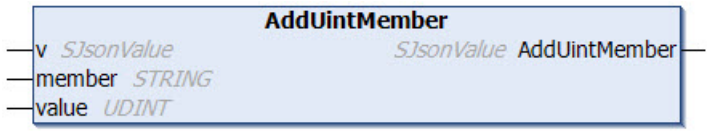
/ Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

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

4.1.16 AddUintMember



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

Syntax

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

Rückgabewert

Name	Typ
AddUintMember	SJsonValue

Eingänge

Name	Typ
v	SJsonValue
value	UDINT

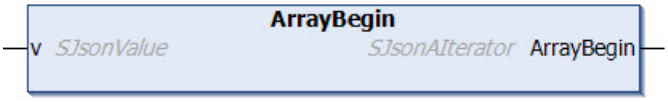
/ Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

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

4.1.17 ArrayBegin



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

Syntax

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

Rückgabewert

Name	Typ
ArrayBegin	SJsonAIterator

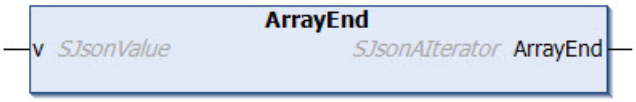
Eingänge

Name	Typ
v	SJsonValue

Beispielaufruf:

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

4.1.18 ArrayEnd



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

Syntax

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

Rückgabewert

Name	Typ
ArrayEnd	SJsonAIterator

Eingänge

Name	Typ
v	SJsonValue

Beispielaufruf:

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

4.1.19 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
i	SJsonAIterator

Beispielaufruf:

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

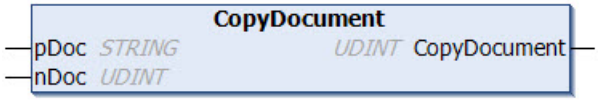
```
sMessage := '{"serialNumber":"123","batteryVoltage":"1547mV","clickType":"SINGLE", "array":
["Hello",2,3]}';
```

Die 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
```

4.1.20 CopyDocument



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

Syntax

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

Rückgabewert

Name	Typ
CopyDocument	UDINT

Eingänge

Name	Typ
nDoc	DINT

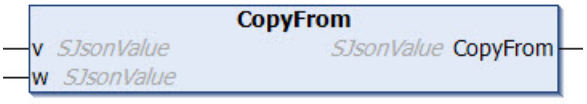
/ Ein-/Ausgänge

Name	Typ
pDoc	STRING

Beispielaufruf:

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

4.1.21 CopyFrom



Syntax

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

Rückgabewert

Name	Typ
CopyFrom	SJsonValue

Eingänge

Name	Typ
v	SJsonValue
w	SJsonValue

4.1.22 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

/ 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"}
```

4.1.23 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

/ 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
```

4.1.24 ExceptionRaised

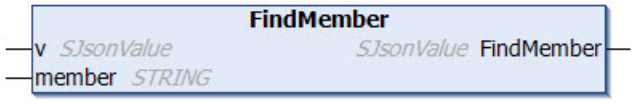
Syntax

```
METHOD ExceptionRaised : BOOL
```

Rückgabewert

Name	Typ
ExceptionRaised	BOOL

4.1.25 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

Eingänge

Name	Typ
v	SJsonValue

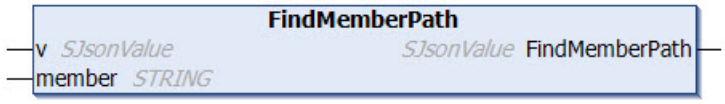
/ Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

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

4.1.26 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

Eingänge

Name	Typ
v	SJsonValue

/ 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
    }
  },
]
```

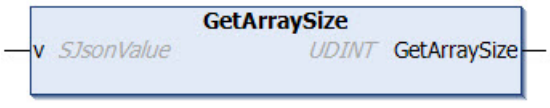
Der Beispielaufruf greift auf das zweite Element des äußeren Arrays zu (#1), anschließend auf das dritte Element des Arrays unter dem Unterelement *Third*.

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

4.1.27 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

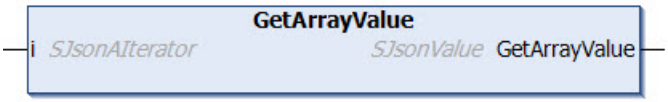
Eingänge

Name	Typ
v	SJsonValue

Beispielaufruf:

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

4.1.28 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

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
```

4.1.29 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

Eingänge

Name	Typ
v	SJsonValue
idx	UDINT

Beispielaufruf:

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

4.1.30 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

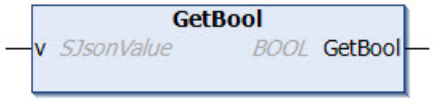
Eingänge

Name	Typ
v	SJsonValue
p	PVOID
n	DINT

Beispielaufruf:

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

4.1.31 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

4.1.32 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
```

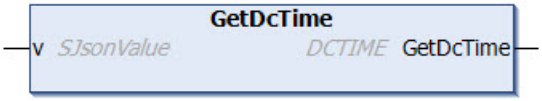
Rückgabewert

Name	Typ
GetDateTime	DATE_AND_TIME

Eingänge

Name	Typ
v	SJsonValue

4.1.33 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

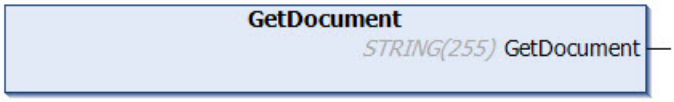
Eingänge

Name	Typ
v	SJsonValue

Beispielaufruf:

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

4.1.34 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 [▶ 37]() verwendet werden.

Syntax

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

Rückgabewert

Name	Typ
GetDocument	STRING(255)

Beispielaufruf:

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

4.1.35 GetDocumentLength

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

Syntax

```
METHOD GetDocumentLength: UDINT
```

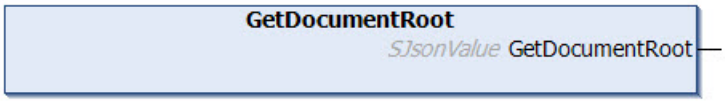
Rückgabewert

Name	Typ
GetDocumentLength	UDINT

Beispielaufruf:

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

4.1.36 GetDocumentRoot



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

Syntax

```
METHOD GetDocumentRoot : SJsonValue
```

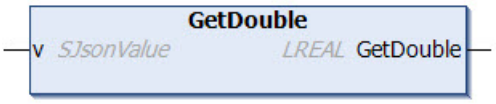
Rückgabewert

Name	Typ
GetDocumentRoot	SJsonValue

Beispielaufruf:

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

4.1.37 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

4.1.38 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

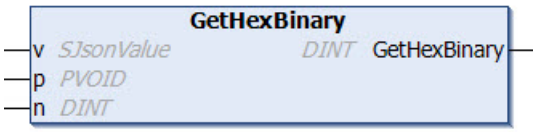
Eingänge

Name	Typ
v	SJsonValue

Beispielaufruf:

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

4.1.39 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
p	PVOID
n	DINT

Beispielaufruf:

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

4.1.40 GetInt

Diese Methode liefert den Value eines Properties vom Datentyp DINT.

Syntax

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

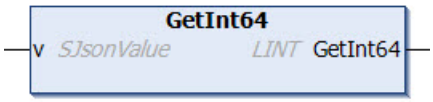
Rückgabewert

Name	Typ
GetInt	DINT

Eingänge

Name	Typ
v	SJsonValue

4.1.41 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

4.1.42 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 [▶ 38]() verwendet werden.

Syntax

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

Rückgabewert

Name	Typ
GetJson	STRING(255)

Eingänge

Name	Typ
v	SJsonValue

Beispielaufruf:

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

4.1.43 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

Beispielaufruf:

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

4.1.44 GetMaxDecimalPlaces

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

Syntax

```
METHOD GetMaxDecimalPlaces : DINT
```

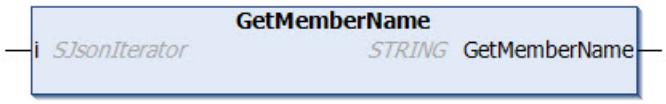
Rückgabewert

Name	Typ
GetMaxDecimalPlaces	DINT

Beispielaufruf:

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

4.1.45 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)

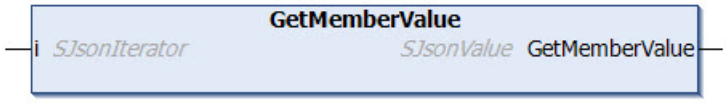
Eingänge

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
```

4.1.46 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

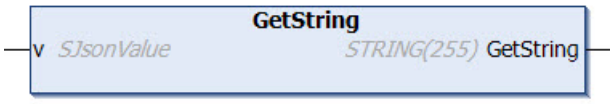
Eingänge

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
```

4.1.47 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 [▶ 39]() verwendet werden.

Syntax

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

Rückgabewert

Name	Typ
GetString	STRING(255)

Eingänge

Name	Typ
v	SJsonValue

4.1.48 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

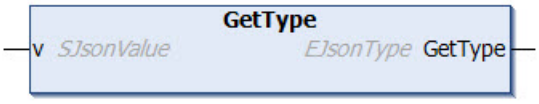
Eingänge

Name	Typ
v	SJsonValue

Beispielaufruf:

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

4.1.49 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

Beispielaufruf:

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

4.1.50 GetUint

Diese Methode liefert den Value eines Properties vom Datentyp UDINT.

Syntax

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

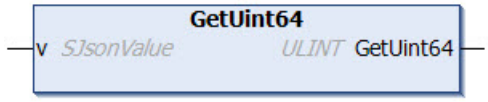
Rückgabewert

Name	Typ
GetUint	UDINT

Eingänge

Name	Typ
v	SJsonValue

4.1.51 GetUint64



Diese Methode liefert den Value eines Properties vom Datentyp ULINT.

Syntax

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

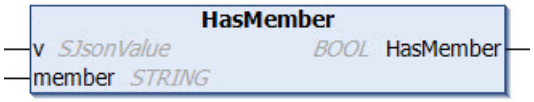
Rückgabewert

Name	Typ
GetUint64	ULINT

Eingänge

Name	Typ
v	SJsonValue

4.1.52 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

/ Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

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

4.1.53 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

4.1.54 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

4.1.55 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
```

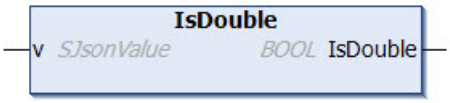
Rückgabewert

Name	Typ
IsBool	BOOL

Eingänge

Name	Typ
v	SJsonValue

4.1.56 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

4.1.57 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
```

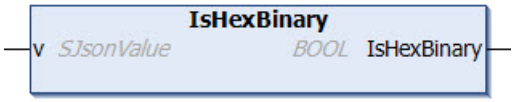
Rückgabewert

Name	Typ
IsFalse	BOOL

Eingänge

Name	Typ
v	SJsonValue

4.1.58 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

Beispielaufruf:

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

4.1.59 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

4.1.60 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
```

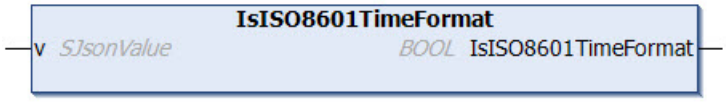
Rückgabewert

Name	Typ
IsInt64	BOOL

Eingänge

Name	Typ
v	SJsonValue

4.1.61 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

4.1.62 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
```

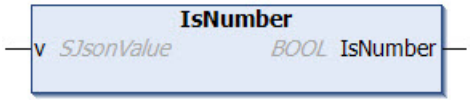
Rückgabewert

Name	Typ
IsNull	BOOL

Eingänge

Name	Typ
v	SJsonValue

4.1.63 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

4.1.64 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

4.1.65 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

4.1.66 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

4.1.67 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

4.1.68 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
```

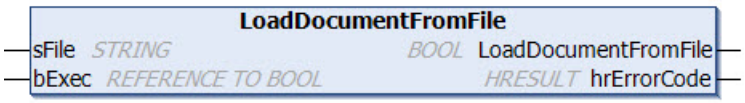
Rückgabewert

Name	Typ
IsUint64	BOOL

Eingänge

Name	Typ
v	SJsonValue

4.1.69 LoadDocumentFromFile



Diese Methode lädt ein JSON-Dokument aus einer Datei.

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

Syntax

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

Rückgabewert

Name	Typ
LoadDocumentFromFile	BOOL

Eingänge

Name	Typ
bExec	REFERENCE TO BOOL

/ Ein-/Ausgänge

Name	Typ
sFile	STRING

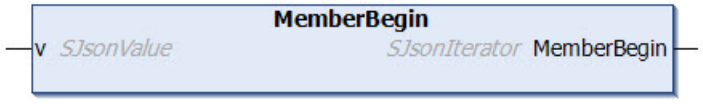
Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

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

4.1.70 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

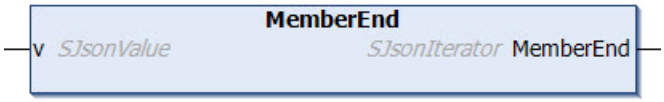
Eingänge

Name	Typ
v	SJsonValue

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
```

4.1.71 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

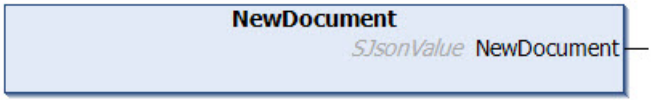
Eingänge

Name	Typ
v	SJsonValue

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
```

4.1.72 NewDocument



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

Syntax

```
METHOD NewDocument : SJsonValue
```

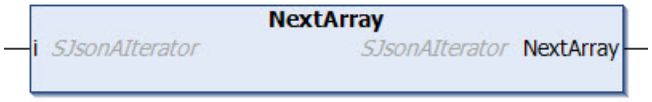
Beispielaufruf:

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

Rückgabewert

Name	Typ
NewDocument	SJsonValue

4.1.73 NextArray



Syntax

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

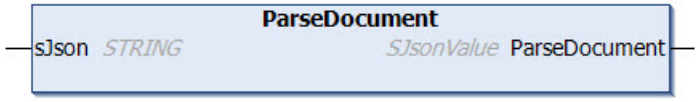
Rückgabewert

Name	Typ
NextArray	SJsonAIterator

Eingänge

Name	Typ
i	SJsonAIterator

4.1.74 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

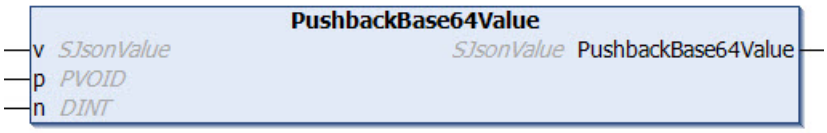
Eingänge

Name	Typ
sJson	STRING

Beispielaufruf:

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

4.1.75 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

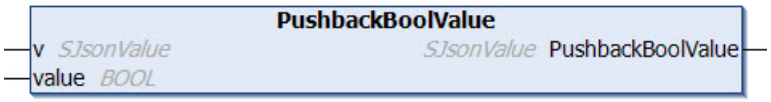
Eingänge

Name	Typ
v	sJsonValue
p	PVOID
n	DINT

Beispielaufruf:

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

4.1.76 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

Eingänge

Name	Typ
v	sJsonValue
value	BOOL

Beispielaufruf:

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

4.1.77 PushbackCopyValue

Diese Methode fügt ein JSON-Dokument aus dem Speicher eines FB_JsonDomParser [▶ 19] zum Ende eines Arrays hinzu.

Syntax

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

Rückgabewert

Name	Typ
PushbackCopyValue	SJsonValue

Eingänge

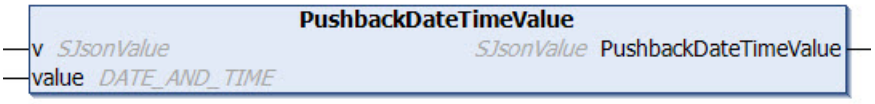
Name	Typ
v	SJsonValue
json	SJsonValue

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);
```

4.1.78 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

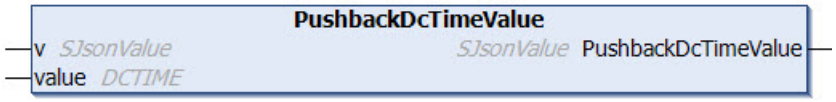
Eingänge

Name	Typ
v	sJsonValue
value	DATE_AND_TIME

Beispielaufruf:

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

4.1.79 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

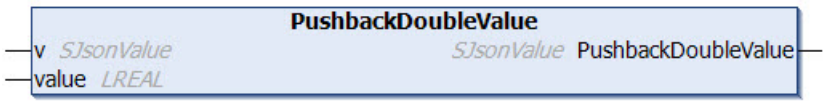
Eingänge

Name	Typ
v	sJsonValue
value	DCTIME

Beispielaufruf:

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

4.1.80 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

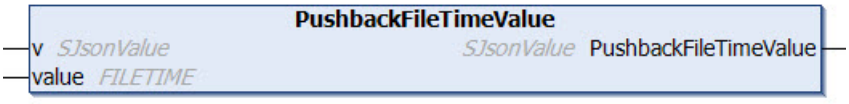
Eingänge

Name	Typ
v	sJsonValue
value	LREAL

Beispielaufruf:

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

4.1.81 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

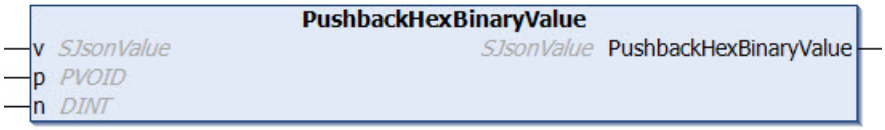
Eingänge

Name	Typ
v	sJsonValue
value	FILETIME

Beispielaufruf:

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

4.1.82 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

Eingänge

Name	Typ
v	sJsonValue
p	PVOID
n	DINT

Beispielaufruf:

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

4.1.83 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

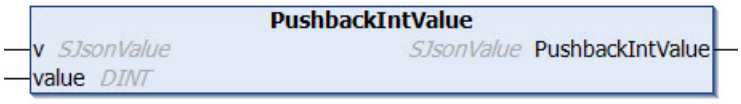
Eingänge

Name	Typ
v	sJsonValue
value	LINT

Beispielaufruf:

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

4.1.84 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

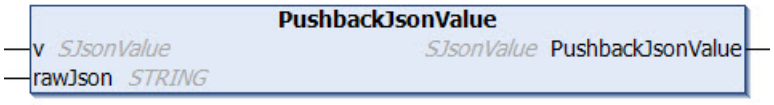
Eingänge

Name	Typ
v	sJsonValue
value	DINT

Beispielaufruf:

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

4.1.85 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

Eingänge

Name	Typ
v	SJsonValue

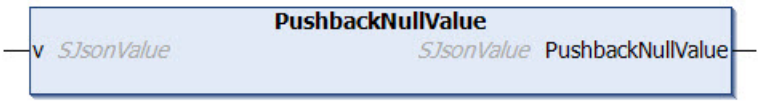
/ Ein-/Ausgänge

Name	Typ
rawJson	STRING

Beispielaufruf:

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

4.1.86 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

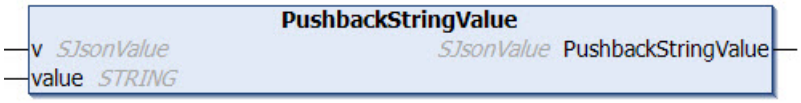
Eingänge

Name	Typ
v	SJsonValue

Beispielaufruf:

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

4.1.87 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

Eingänge

Name	Typ
v	SJsonValue

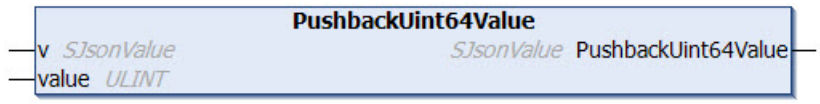
Ein-/Ausgänge

Name	Typ
value	STRING

Beispielaufruf:

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

4.1.88 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

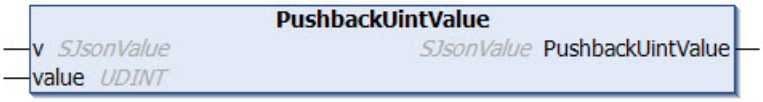
Eingänge

Name	Typ
v	SJsonValue
value	ULINT

Beispielaufruf:

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

4.1.89 PushbackUintValue



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

Syntax

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

Rückgabewert

Name	Typ
PushbackUintValue	SJsonValue

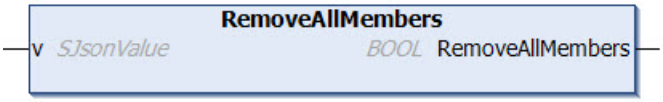
Eingänge

Name	Typ
v	SJsonValue
value	UDINT

Beispielaufruf:

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

4.1.90 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

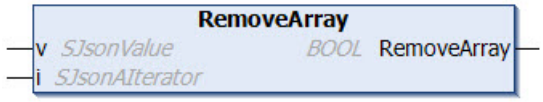
Eingänge

Name	Typ
v	SJsonValue

Beispielaufruf:

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

4.1.91 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
i	SJsonAIterator

Beispielaufruf:

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

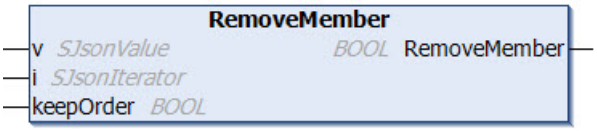
```
sMessage := '{"serialNumber":"123","batteryVoltage":"1547mV","clickType":"SINGLE", "array":
["Hello",2,3]}';
```

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
```

4.1.92 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
i	SJsonIterator
keepOrder	BOOL

Beispielaufruf:

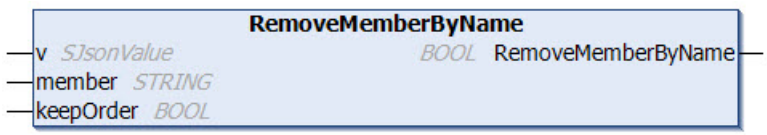
Gegeben sei das folgende JSON Dokument, das in den DOM-Speicher geladen wird:

```
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
```

4.1.93 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
keepOrder	BOOL

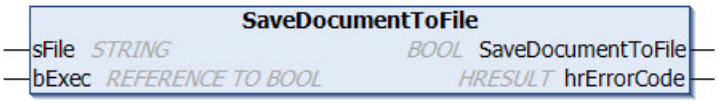
/ Ein-/Ausgänge

Name	Typ
member	STRING

Beispielaufruf:

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

4.1.94 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

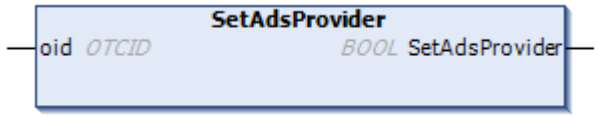
Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

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

4.1.95 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.

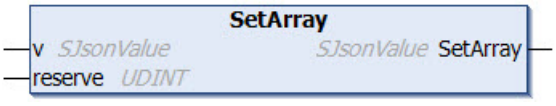
Eingänge

Name	Typ	Beschreibung
oid	OTCID	Object ID der Task.

Beispielaufruf:

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

4.1.96 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

Eingänge

Name	Typ
v	SJsonValue

Beispielaufruf:

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

4.1.97 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

Eingänge

Name	Typ
v	SJsonValue
p	PVOID
n	DINT

Beispielaufruf:

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

4.1.98 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

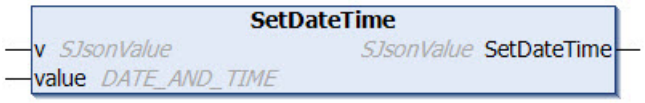
Eingänge

Name	Typ
v	SJsonValue
value	BOOL

Beispielaufruf:

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

4.1.99 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

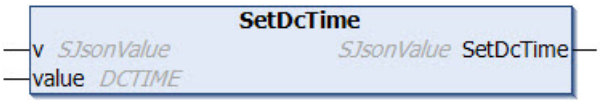
Eingänge

Name	Typ
v	SJsonValue
value	DATE_AND_TIME

Beispielaufruf:

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

4.1.100 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

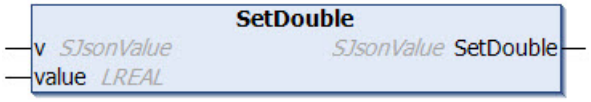
Eingänge

Name	Typ
v	SJsonValue
value	DCTIME

Beispielaufruf:

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

4.1.101 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

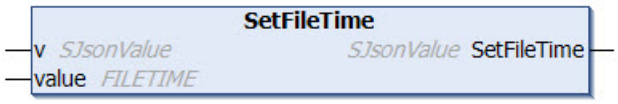
Eingänge

Name	Typ
v	SJsonValue
value	LREAL

Beispielaufruf:

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

4.1.102 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

Eingänge

Name	Typ
v	SJsonValue
value	FILETIME

Beispielaufruf:

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

4.1.103 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

Eingänge

Name	Typ
v	SJsonValue
p	PVOID
n	DINT

Beispielaufruf:

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

4.1.104 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

Eingänge

Name	Typ
v	SJsonValue
value	DINT

Beispielaufruf:

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

4.1.105 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

Eingänge

Name	Typ
v	SJsonValue
value	LINT

Beispielaufruf:

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

4.1.106 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

Eingänge

Name	Typ
v	SJsonValue

/ Ein-/Ausgänge

Name	Typ
rawJson	STRING

Beispielaufruf:

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

4.1.107 SetMaxDecimalPlaces

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

Syntax

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

Eingänge

Name	Typ
dp	DINT

Beispielaufruf:

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

4.1.108 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

Eingänge

Name	Typ
v	SJsonValue

Beispielaufruf:

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

4.1.109 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

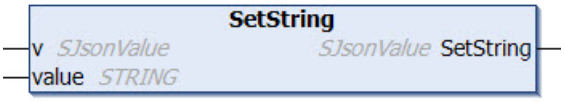
Eingänge

Name	Typ
v	SJsonValue

Beispielaufruf:

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

4.1.110 SetString



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

Syntax

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

Rückgabewert

Name	Typ
SetString	SJsonValue

Eingänge

Name	Typ
v	SJsonValue

/ Ein-/Ausgänge

Name	Typ
value	STRING

Beispielaufruf:

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

4.1.111 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

Eingänge

Name	Typ
v	SJsonValue
value	UDINT

Beispielaufruf:

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

4.1.112 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

Eingänge

Name	Typ
v	SJsonValue
value	ULINT

Beispielaufruf:

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

4.1.113 Swap

Syntax

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

Rückgabewert

Name	Typ
Swap	BOOL

Eingänge

Name	Typ
v	SJsonValue
w	SJsonValue

4.2 FB_JsonDynDomParser

Dieser Funktionsblock ist von demselben internen Funktionsbaustein abgeleitet wie der FB_JsonDomParser [▶ 19] 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 [▶ 19] eine bessere Performance ermöglicht.

Strings im UTF-8-Format

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

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

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

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

Syntax

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

Ausgänge

Name	Typ
initStatus	HRESULT

Methoden

Alle Methoden finden Sie in FB_JsonDomParser [▶ 19].

Voraussetzungen

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

4.3 FB_JsonSaxReader

Syntax

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

Ausgänge

Name	Typ
initStatus	HRESULT

Methoden

Name	Beschreibung
DecodeBase64 [▶ 90]	Konvertiert einen Base64-formatierten String in Binärdaten.
DecodeDateTime [▶ 91]	Erzeugt aus einem genormten ISO8601-Zeitformat (z. B. YYYY-MM-DDThh:mm:ss) eine SPS-Variable vom Datentyp DATE_AND_TIME bzw. DT.
DecodeDcTime [▶ 91]	Erzeugt aus einem genormten ISO8601 Zeitformat (z. B. YYYY-MM-DDThh:mm:ss) eine SPS-Variable vom Datentyp DCTIME.
DecodeFileTime [▶ 92]	Erzeugt aus einem genormten ISO8601-Zeitformat (z. B. YYYY-MM-DDThh:mm:ss) eine SPS-Variable vom Datentyp FILETIME.
DecodeHexBinary [▶ 93]	Konvertiert einen String, der Hexadezimalwerte enthält, in Binärdaten um.
IsBase64 [▶ 94]	Prüft, ob der übergebene String dem Base64-Format entspricht.
IsHexBinary [▶ 94]	Prüft, ob der übergebene String aus Hexadezimalwerten besteht.
IsISO8601TimeFormat [▶ 95]	Prüft, ob der übergebene String dem genormten ISO8601-Zeitformat entspricht.
Parse [▶ 95]	Startet den SAX-Reader-Parsing-Vorgang.
ParseValues [▶ 96]	Startet den SAX-Reader-Parsing-Vorgang.

Voraussetzungen

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

4.3.1 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);
```

4.3.2 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);
```

4.3.3 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);
```

4.3.4 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
```

Rückgabewert

Name	Typ
DecodeDateTime	BOOL

/ Ein-/Ausgänge

Name	Typ
sFT	STRING

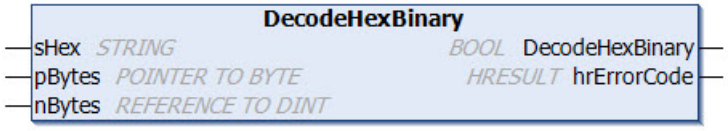
Ausgänge

Name	Typ
nFT	FILETIME
hrErrorCode	HRESULT

Beispielaufruf:

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

4.3.5 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

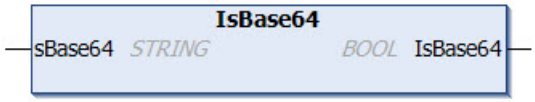
Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

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

4.3.6 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

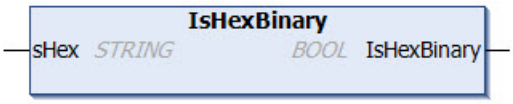
/ Ein-/Ausgänge

Name	Typ
sBase64	STRING

Beispielaufruf:

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

4.3.7 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

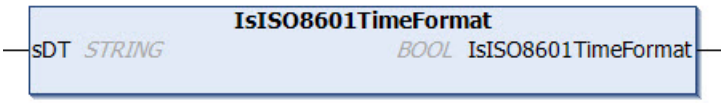
/ Ein-/Ausgänge

Name	Typ
sHex	STRING

Beispielaufruf:

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

4.3.8 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');
```

4.3.9 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

4.3.10 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

4.4 FB_JsonSaxWriter

Strings im UTF-8-Format

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

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

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

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

Syntax

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

Ausgänge

Name	Typ
initStatus	HRESULT

Methoden

Name	Beschreibung
AddBase64 [▶ 100]	Fügt einen Wert vom Datentyp Base64 zu einem Property hinzu.
AddBool [▶ 100]	Fügt einen Wert vom Datentyp BOOL zu einem Property hinzu.
AddDateTime [▶ 101]	Fügt einen Wert vom Datentyp DATE_AND_TIME zu einem Property hinzu.
AddDcTime [▶ 101]	Fügt einen Wert vom Datentyp DCTIME zu einem Property hinzu.
AddDint [▶ 101]	Fügt einen Wert vom Datentyp DINT zu einem Property hinzu.
AddFileTime [▶ 102]	Fügt einen Wert vom Datentyp FILETIME zu einem Property hinzu.
AddHexBinary [▶ 102]	Fügt einen HexBinary-Wert zu einem Property hinzu.
AddKey [▶ 103]	Fügt einen neuen Property-Key an der aktuellen Position des SAX Writers hinzu.
AddKeyBool [▶ 103]	Legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp BOOL an.
AddKeyDateTime [▶ 104]	Legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp DATE_AND_TIME an.
AddKeyDcTime [▶ 104]	Legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp DCTIME an.
AddKeyFileTime [▶ 105]	Legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp FILETIME an.
AddKeyLreal [▶ 105]	Legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp LREAL an.
AddKeyNull [▶ 106]	Legt einen neuen Property-Key an und initialisiert dessen Wert mit null.
AddKeyNumber [▶ 106]	Legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp DINT an.
AddKeyString [▶ 107]	Legt einen neuen Property-Key und gleichzeitig einen Wert vom Datentyp STRING an.
AddLint [▶ 107]	Fügt einen Wert vom Datentyp LINT zu einem Property hinzu.
AddLreal [▶ 108]	Fügt einen Wert vom Datentyp LREAL zu einem Property hinzu.
AddNull [▶ 108]	Fügt den Wert null zu einem Property hinzu.
AddRawArray [▶ 108]	Fügt ein gültiges JSON-Array zu einem gegebenen Property als Value hinzu.
AddRawObject [▶ 109]	Fügt ein gültiges JSON-Objekt zu einem gegebenen Property als Value hinzu.
AddReal [▶ 109]	Fügt einen Wert vom Datentyp REAL zu einem Property hinzu.
AddString [▶ 110]	Fügt einen Wert vom Datentyp STRING zu einem Property hinzu.
AddUdint [▶ 110]	Fügt einen Wert vom Datentyp UDINT zu einem Property hinzu.
AddUlint [▶ 111]	Fügt einen Wert vom Datentyp ULINT zu einem Property hinzu.
CopyDocument [▶ 111]	Kopiert den Inhalt des aktuell mit dem SAX Writer erstellten JSON-Objekts in eine Zielvariable vom Datentyp STRING.
EndArray [▶ 112]	Erzeugt den Abschluss eines angefangenen JSON-Arrays und fügt sie an der aktuellen Position des SAX Writers ein.
EndObject [▶ 112]	Erzeugt den Abschluss eines angefangenen JSON-Objekts und fügt sie an der aktuellen Position des SAX Writers ein.
GetDocument [▶ 113]	Liefert den Inhalt des aktuell mit dem SAX Writer erstellten JSON-Objekts und gibt diesen als Datentyp STRING(255) zurück.
GetDocumentLength [▶ 113]	Liefert die Länge des aktuell mit dem SAX Writer erstellten JSON-Objekts und gibt diese als Datentyp UDINT zurück.
GetMaxDecimalPlaces [▶ 114]	
IsComplete	
ResetDocument [▶ 114]	Setzt das aktuell mit dem SAX Writer erstellte JSON-Objekt zurück.
SetMaxDecimalPlaces [▶ 114]	

Name	Beschreibung
StartArray [▶ 115]	Erzeugt den Anfang eines neuen JSON-Arrays und fügt sie an der aktuellen Position des SAX Writers ein.
StartObject [▶ 115]	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

4.4.1 AddBase64

Diese Methode fügt einen Wert vom Datentyp Base64 zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 103] ein entsprechendes Property erstellt.

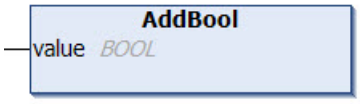
Syntax

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

Eingänge

Name	Typ
pBytes	POINTER TO BYTE
nBytes	DINT

4.4.2 AddBool



Diese Methode fügt einen Wert vom Datentyp BOOL zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 103] 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);
```

4.4.3 AddDateTime

Diese Methode fügt einen Wert vom Datentyp DATE_AND_TIME zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 103] 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
```

4.4.4 AddDcTime

Diese Methode fügt einen Wert vom Datentyp DCTIME zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 103] 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
```

4.4.5 AddDint

Diese Methode fügt einen Wert vom Datentyp DINT zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 103] ein entsprechendes Property erstellt.

Syntax

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

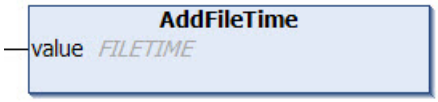
Eingänge

Name	Typ
value	DINT

Beispielaufruf:

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

4.4.6 AddFileTime



Diese Methode fügt einen Wert vom Datentyp FILETIME zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 103] 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
```

4.4.7 AddHexBinary

Diese Methode fügt einen HexBinary-Wert zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 103] ein entsprechendes Property erstellt.

Syntax

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

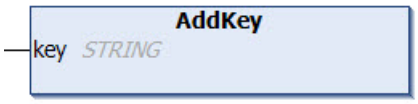
Eingänge

Name	Typ
pBytes	POINTER TO BYTE
nBytes	DINT

Beispielaufruf:

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

4.4.8 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 [▶ 100], AddBool [▶ 100], AddDateTime [▶ 101], AddDcTime [▶ 101], AddDint [▶ 101], AddFileTime [▶ 102], AddHexBinary [▶ 102], AddLint [▶ 107], AddLreal [▶ 108], AddNull [▶ 108], AddRawArray [▶ 108], AddRawObject [▶ 109], AddReal [▶ 109], AddString [▶ 110], AddUdint [▶ 110], AddUlint [▶ 111].

Syntax

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

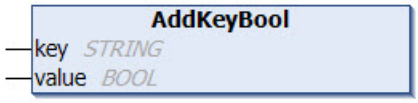
/ Ein-/Ausgänge

Name	Typ
key	STRING

Beispielaufruf:

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

4.4.9 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

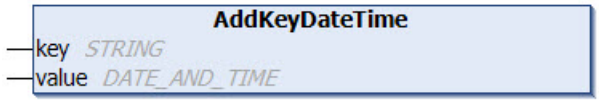
/ Ein-/Ausgänge

Name	Typ
key	STRING

Beispielaufruf:

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

4.4.10 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

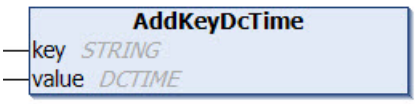
/ Ein-/Ausgänge

Name	Typ
key	STRING

Beispielaufruf:

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

4.4.11 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

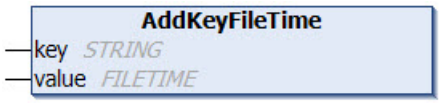
/ Ein-/Ausgänge

Name	Typ
key	STRING

Beispielaufruf:

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

4.4.12 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

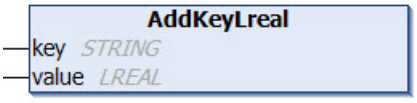
/ Ein-/Ausgänge

Name	Typ
key	STRING

Beispielaufruf:

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

4.4.13 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

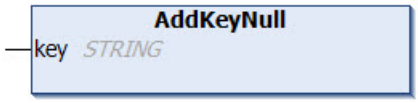
/ Ein-/Ausgänge

Name	Typ
key	STRING

Beispielaufruf:

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

4.4.14 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
```

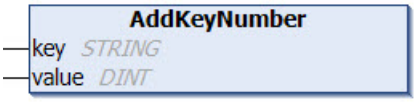
/ Ein-/Ausgänge

Name	Typ
key	STRING

Beispielaufruf:

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

4.4.15 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

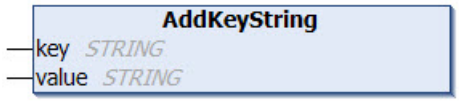
/ Ein-/Ausgänge

Name	Typ
key	STRING

Beispielaufruf:

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

4.4.16 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');
```

4.4.17 AddLint

Diese Methode fügt einen Wert vom Datentyp LINT zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 103] 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);
```

4.4.18 AddLreal

Diese Methode fügt einen Wert vom Datentyp LREAL zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 103] 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);
```

4.4.19 AddNull

Diese Methode fügt den Wert „null“ zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 103] ein entsprechendes Property erstellt.

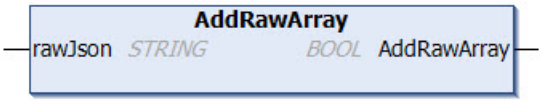
Syntax

```
METHOD AddNull
```

Beispielaufruf:

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

4.4.20 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() [▶ 103], StartArray() [▶ 115] oder als erster Aufruf nach einem ResetDocument() [▶ 114].

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');
```

4.4.21 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() [▶ 103], StartArray() [▶ 115] oder als erster Aufruf nach einem ResetDocument() [▶ 114].

Syntax

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

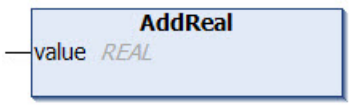
/ Ein-/Ausgänge

Name	Typ
rawJson	STRING

Beispielaufruf:

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

4.4.22 AddReal



Diese Methode fügt einen Wert vom Datentyp REAL zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 103] 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);
```

4.4.23 AddString

Diese Methode fügt einen Wert vom Datentyp STRING zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 103] 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');
```

4.4.24 AddUdint

Diese Methode fügt einen Wert vom Datentyp UDINT zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 103] 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);
```

4.4.25 AddUlint

Diese Methode fügt einen Wert vom Datentyp ULINT zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 103] ein entsprechendes Property erstellt.

Syntax

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

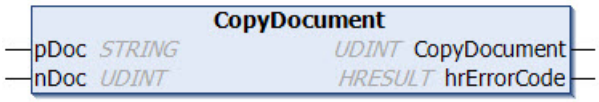
Eingänge

Name	Typ
value	ULINT

Beispielaufruf:

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

4.4.26 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));
```

4.4.27 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();
```

4.4.28 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();
```

4.4.29 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 [▶ 111]() 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();
```

4.4.30 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();
```

4.4.31 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.

4.4.32 ResetDocument

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

Syntax

```
METHOD ResetDocument : HRESULT
```

Rückgabewert

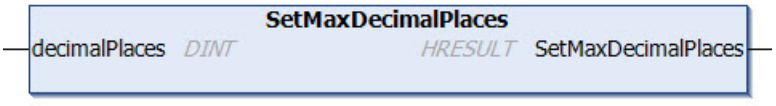
Name	Typ
ResetDocument	HRESULT

Beispielaufruf:

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

4.4.33 SetMaxDecimalPlaces

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



Syntax

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

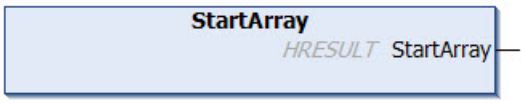
Rückgabewert

Name	Typ
SetMaxDecimalPlaces	HRESULT

Eingänge

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

4.4.34 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();
```

4.4.35 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();
```

4.5 FB_JsonSaxPrettyWriter

Dieser Funktionsbaustein hat einen Unterschied zum FB_JsonSaxWriter [▶ 97]: 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 [▶ 131]	Passt die allgemeinen Formatierungsoptionen für eine Instanz des [<=>] FB_JsonSaxPrettyWriter an.
SetIndent [▶ 131]	Passt die Einrückung benutzerdefiniert an.

Alle weiteren Methoden finden Sie beim FB_JsonSaxWriter [▶ 97].

Voraussetzungen

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

4.5.1 AddBase64

Diese Methode fügt einen Wert vom Datentyp Base64 zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 119] ein entsprechendes Property erstellt.

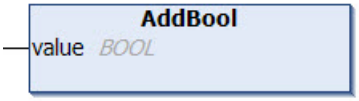
Syntax

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

Eingänge

Name	Typ
pBytes	POINTER TO BYTE
nBytes	DINT

4.5.2 AddBool



Diese Methode fügt einen Wert vom Datentyp BOOL zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 119] 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);
```

4.5.3 AddDateTime

Diese Methode fügt einen Wert vom Datentyp DATE_AND_TIME zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 119] ein entsprechendes Property erstellt.

Syntax

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

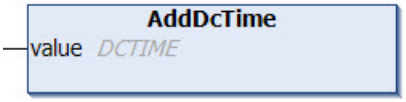
Eingänge

Name	Typ
value	DATE_AND_TIME

Beispielaufruf:

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

4.5.4 AddDcTime



Diese Methode fügt einen Wert vom Datentyp DCTIME zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 119] 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
```

4.5.5 AddDint

Diese Methode fügt einen Wert vom Datentyp DINT zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 119] ein entsprechendes Property erstellt.

Syntax

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

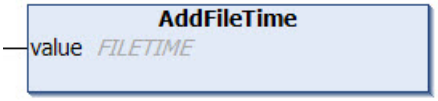
Eingänge

Name	Typ
value	DINT

Beispielaufruf:

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

4.5.6 AddFileTime



Diese Methode fügt einen Wert vom Datentyp FILETIME zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 119] 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
```

4.5.7 AddHexBinary

Diese Methode fügt einen HexBinary-Wert zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 119] ein entsprechendes Property erstellt.

Syntax

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

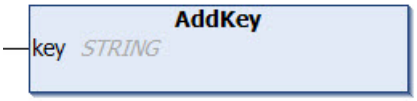
Eingänge

Name	Typ
pBytes	POINTER TO BYTE
nBytes	DINT

Beispielaufruf:

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

4.5.8 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 [▶ 116], AddBool [▶ 117], AddDateTime [▶ 117], AddDcTime [▶ 118], AddDint [▶ 118], AddFileTime [▶ 118], AddHexBinary [▶ 119], AddLint [▶ 124], AddLreal [▶ 124], AddNull [▶ 125], AddRawArray [▶ 125], AddRawObject [▶ 125], AddReal [▶ 126], AddString [▶ 126], AddUdint [▶ 127], AddUlint [▶ 127].

Syntax

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

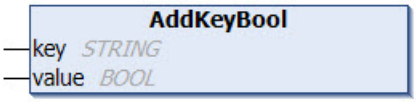
/ Ein-/Ausgänge

Name	Typ
key	STRING

Beispielaufruf:

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

4.5.9 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

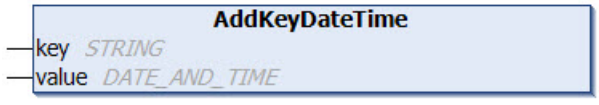
/ Ein-/Ausgänge

Name	Typ
key	STRING

Beispielaufruf:

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

4.5.10 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

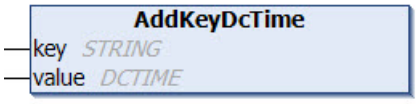
/ Ein-/Ausgänge

Name	Typ
key	STRING

Beispielaufruf:

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

4.5.11 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

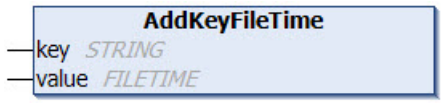
/ Ein-/Ausgänge

Name	Typ
key	STRING

Beispielaufruf:

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

4.5.12 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

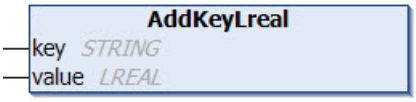
/ Ein-/Ausgänge

Name	Typ
key	STRING

Beispielaufruf:

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

4.5.13 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

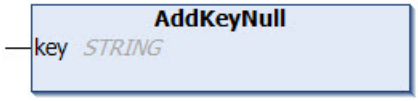
/ Ein-/Ausgänge

Name	Typ
key	STRING

Beispielaufruf:

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

4.5.14 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
```

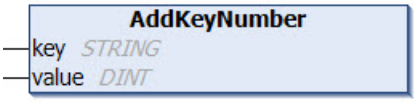
/ Ein-/Ausgänge

Name	Typ
key	STRING

Beispielaufruf:

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

4.5.15 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

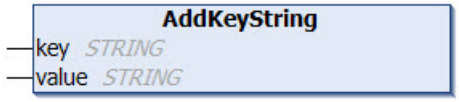
/ Ein-/Ausgänge

Name	Typ
key	STRING

Beispielaufruf:

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

4.5.16 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');
```

4.5.17 AddLint

Diese Methode fügt einen Wert vom Datentyp LINT zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 119] 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);
```

4.5.18 AddLreal

Diese Methode fügt einen Wert vom Datentyp LREAL zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 119] 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);
```

4.5.19 AddNull

Diese Methode fügt den Wert „null“ zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 119] ein entsprechendes Property erstellt.

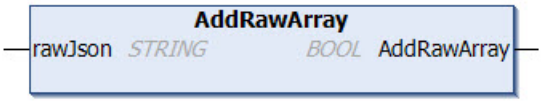
Syntax

```
METHOD AddNull
```

Beispielaufruf:

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

4.5.20 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() [▶ 119], StartArray() [▶ 132] oder als erster Aufruf nach einem ResetDocument() [▶ 130].

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');
```

4.5.21 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() [▶ 119], StartArray() [▶ 132] oder als erster Aufruf nach einem ResetDocument() [▶ 130].

Syntax

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

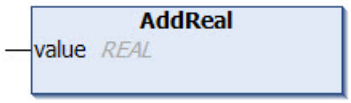
/ Ein-/Ausgänge

Name	Typ
rawJson	STRING

Beispielaufruf:

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

4.5.22 AddReal



Diese Methode fügt einen Wert vom Datentyp REAL zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 119] 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);
```

4.5.23 AddString

Diese Methode fügt einen Wert vom Datentyp STRING zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 119] 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');
```

4.5.24 AddUdint

Diese Methode fügt einen Wert vom Datentyp UDINT zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 119] 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);
```

4.5.25 AddUlint

Diese Methode fügt einen Wert vom Datentyp ULINT zu einem Property hinzu. Üblicherweise wurde vorher mit der Methode AddKey() [▶ 119] ein entsprechendes Property erstellt.

Syntax

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

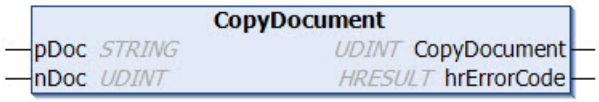
Eingänge

Name	Typ
value	ULINT

Beispielaufruf:

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

4.5.26 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));
```

4.5.27 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();
```

4.5.28 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();
```

4.5.29 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 [▶ 128]() 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();
```

4.5.30 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();
```

4.5.31 GetMaxDecimalPlaces

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

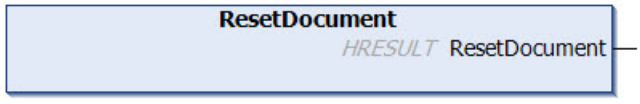
Syntax

```
METHOD GetMaxDecimalPlaces : DINT
```

Rückgabewert

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

4.5.32 ResetDocument



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

Syntax

```
METHOD ResetDocument : HRESULT
```

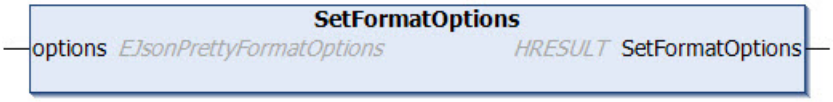
Rückgabewert

Name	Typ
ResetDocument	HRESULT

Beispielaufruf:

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

4.5.33 SetFormatOptions



Diese Methode wird verwendet, um die allgemeinen Formatierungsoptionen für eine Instanz des FB_JsonSaxPrettyWriter [▶ 116] 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);
```

4.5.34 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);
```

4.5.35 SetMaxDecimalPlaces

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

Syntax

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

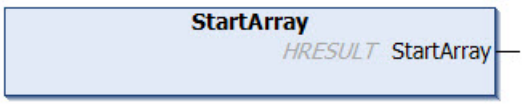
Rückgabewert

Name	Typ
SetMaxDecimalPlaces	HRESULT

Eingänge

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

4.5.36 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();
```

4.5.37 StartObject

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

Syntax

```
METHOD StartObject : HRESULT
```

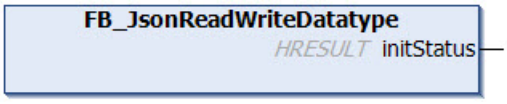
Rückgabewert

Name	Typ
StartObject	HRESULT

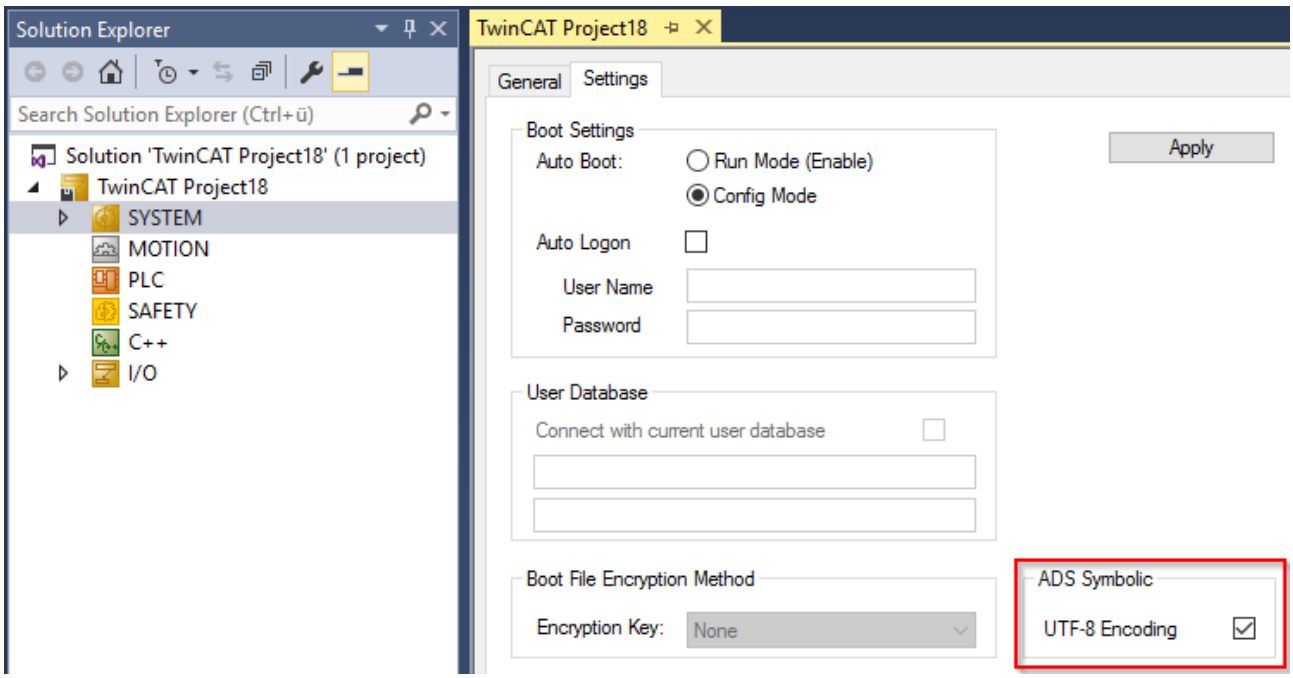
Beispielaufruf:

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

4.6 FB_JsonReadWriteDataType



Zur Verwendung von UTF-8-Zeichen, z. B. bei der automatischen Generierung von Metadaten über den Funktionsbaustein FB_JsonReadWriteDataType [▶ 133], muss im TwinCAT-Projekt das Auswahlkästchen zur Unterstützung von UTF-8 in der Symbolik aktiviert sein. Klicken Sie dazu im Projektbaum doppelt auf **SYSTEM**, öffnen Sie die Registerkarte **Settings** und aktivieren Sie das entsprechende Auswahlkästchen.



Strings im UTF-8-Format

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

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

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

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

Ausgänge

Name	Typ
initStatus	HRESULT

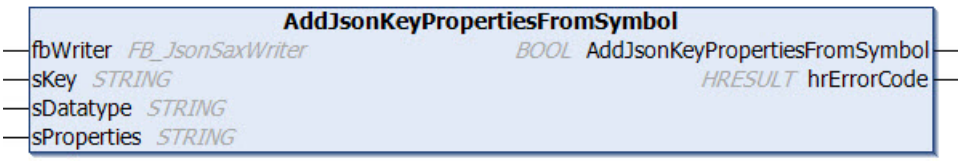
Methoden

Name	Beschreibung
FB_JsonReadWriteDataType	Metadaten werden über SPS-Attribute in die JSON-Repräsentation einer SPS-Datenstruktur auf einem FB_JsonSaxWriter-Objekt hinzugefügt.
AddJsonKeyValueFromSymbol [▶ 136]	Erzeugt die JSON-Repräsentation einer SPS-Datenstruktur auf einem FB_JsonSaxWriter-Objekt.
AddJsonValueFromSymbol [▶ 137]	Erzeugt die JSON-Repräsentation einer SPS-Datenstruktur auf einem FB_JsonSaxWriter-Objekt.
CopyJsonStringFromSymbol [▶ 138]	Erzeugt die JSON-Repräsentanz eines Symbols und kopiert diese in eine Variable vom Datentyp STRING.
CopyJsonStringFromSymbolProperties [▶ 139]	Erzeugt eine entsprechende JSON-Repräsentation von SPS-Attributen an einem Symbol.
CopySymbolNameByAddress [▶ 140]	Liefert den vollständigen (ADS-)Symbolnamen eines übergebenen Symbols.
GetDataTypeNameByAddress [▶ 141]	Liefert den Datentypnamen eines übergebenen Symbols.
GetJsonFromSymbol [▶ 142]	Erzeugt die entsprechende JSON-Repräsentation eines Symbols.
GetJsonStringFromSymbol [▶ 143]	Erzeugt die entsprechende JSON-Repräsentation eines Symbols.
GetJsonStringFromSymbolProperties [▶ 144]	Erzeugt eine entsprechende JSON-Repräsentation von SPS-Attributen an einem Symbol.
GetSizeJsonStringFromSymbol [▶ 145]	Liest die Größe der JSON-Repräsentanz eines Symbols aus.
GetSizeJsonStringFromSymbolProperties [▶ 146]	Liest die Größe der JSON-Repräsentanz von SPS-Attributen an einem Symbol aus.
GetSymbolNameByAddress [▶ 146]	Liefert den vollständigen (ADS-)Symbolnamen eines übergebenen Symbols.
SetSymbolFromJson [▶ 147]	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

4.6.1 AddJsonKeyPropertiesFromSymbol



Mit dieser Methode können Metadaten über SPS-Attribute in die JSON-Repräsentation einer SPS-Datenstruktur auf einem FB_JsonSaxWriter [▶ 97]-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:

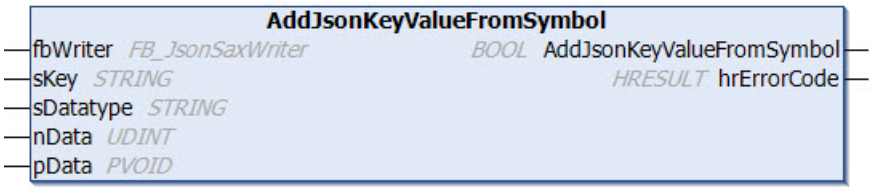
```
{attribute 'Unit'        := 'm/s'}
{attribute 'DisplayName' := 'Speed'}
Sensor1                  : REAL;
```

Ein vollständiges Beispiel zur Verwendung dieser Methode finden Sie im Abschnitt Tc3JsonXmlSampleJsonDataType [▶ 219].

Beispielaufruf:

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

4.6.2 AddJsonKeyValueFromSymbol



Diese Methode erzeugt die JSON-Repräsentation einer SPS-Datenstruktur auf einem FB_JsonSaxWriter [▶ 97]-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() [▶ 137] 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

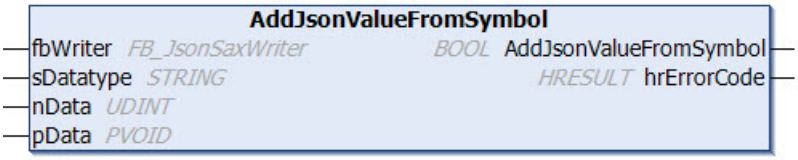
Name	Typ
hrErrorCode	HRESULT

Ein vollständiges Beispiel zur Verwendung dieser Methode finden Sie im Abschnitt Tc3JsonXmlSampleJsonDataType [▶ 219].

Beispielaufruf:

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

4.6.3 AddJsonValueFromSymbol



Diese Methode erzeugt die JSON-Repräsentation einer SPS-Datenstruktur auf einem FB_JsonSaxWriter [▶ 97]-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 [▶ 219].

Beispielaufruf:

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

4.6.4 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

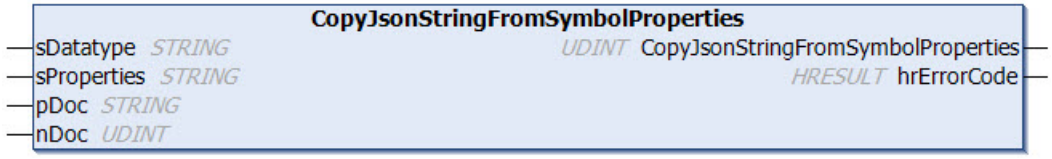
Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

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

4.6.5 CopyJsonStringFromSymbolProperties



Diese Methode erzeugt eine entsprechende JSON-Repräsentation von SPS-Attributen an einem Symbol. Im Unterschied zur Methode AddJsonKeyPropertiesFromSymbol [▶ 135] 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));
```

4.6.6 CopySymbolNameByAddress

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

Syntax

```
METHOD CopySymbolNameByAddress : UDINT
VAR_INPUT
    nData      : UDINT;    // size of symbol
    pData      : PVOID;    // address of symbol
END_VAR
VAR_IN_OUT CONSTANT
```

```
    sName     : STRING;    // target string buffer where the symbol name should be copied to
END_VAR
VAR_INPUT
    nName     : UDINT;    // size in bytes of target string buffer
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

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

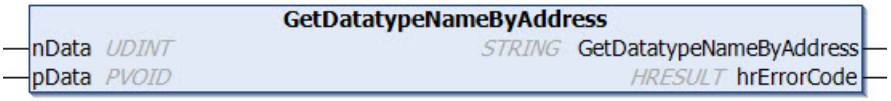
Beispielaufruf:

```
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

4.6.7 GetDataTypeNameByAddress



Diese Methode liefert den Datentypnamen eines übergebenen Symbols.

Syntax

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

Rückgabewert

Name	Typ
GetDataTypeNameByAddress	STRING

Eingänge

Name	Typ
nData	UDINT
pData	PVOID

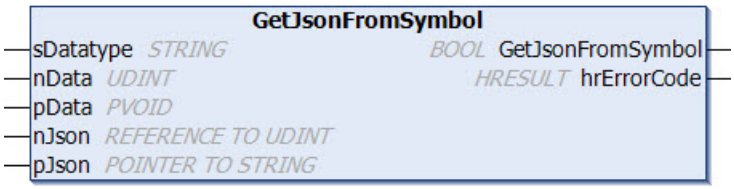
Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

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

4.6.8 GetJsonFromSymbol



Diese Methode erzeugt die entsprechende JSON-Repräsentation eines Symbols. Im Unterschied zur Methode AddJsonValueFromSymbol() [▶ 137] 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

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));
```

4.6.9 GetJsonStringFromSymbol

Diese Methode erzeugt die entsprechende JSON-Repräsentation eines Symbols. Im Unterschied zur Methode AddJsonValueFromSymbol() [▶ 137] 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 [▶ 138]() 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

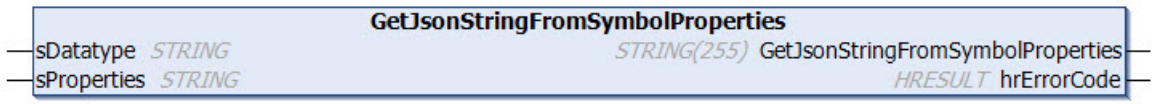
Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

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

4.6.10 GetJsonStringFromSymbolProperties



Diese Methode erzeugt eine entsprechende JSON-Repräsentation von SPS-Attributen an einem Symbol. Im Unterschied zur Methode AddJsonKeyPropertiesFromSymbol [▶ 135] 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 [▶ 139]() 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');
```

4.6.11 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

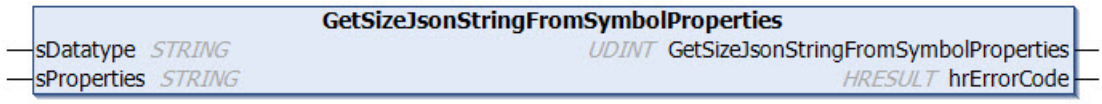
Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

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

4.6.12 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

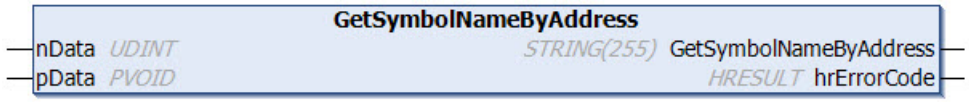
Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

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

4.6.13 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() [▶ 140] 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

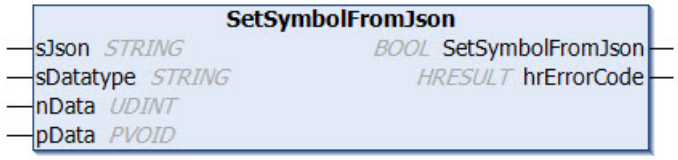
Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

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

4.6.14 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));
```

4.7 FB_XmlDomParser

Strings im UTF-8-Format

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

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

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

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

Ausgänge

Name	Typ
initStatus	HRESULT

Methoden

Name	Beschreibung
AppendAttribute [▶ 155]	Fügt ein neues Attribut zu einem existierenden Knoten hinzu.
AppendAttributeAsBool [▶ 155]	Fügt ein neues Attribut zu einem existierenden Knoten hinzu (Boolean).
AppendAttributeAsDouble [▶ 156]	Fügt ein neues Attribut zu einem existierenden Knoten hinzu (Double).
AppendAttributeAsFloat [▶ 157]	Fügt ein neues Attribut zu einem existierenden Knoten hinzu (Float).
AppendAttributeAsInt [▶ 157]	Fügt ein neues Attribut zu einem existierenden Knoten hinzu (Integer).
AppendAttributeAsLint [▶ 158]	Fügt ein neues Attribut zu einem existierenden Knoten hinzu (Integer64).
AppendAttributeAsUint [▶ 159]	Fügt ein neues Attribut zu einem existierenden Knoten hinzu (Unsigned Integer).
AppendAttributeAsUlint [▶ 160]	Fügt ein neues Attribut zu einem existierenden Knoten hinzu (Unsigned Integer64).
AppendAttributeCopy [▶ 160]	Fügt ein neues Attribut zu einem existierenden Knoten hinzu.
AppendChild [▶ 161]	Fügt einen neuen Knoten unterhalb eines existierenden Knotens ein.
AppendChildAsBool [▶ 162]	Fügt einen neuen Knoten unterhalb eines existierenden Knotens ein (Boolean).
AppendChildAsDouble [▶ 163]	Fügt einen neuen Knoten unterhalb eines existierenden Knotens ein (Double).
AppendChildAsFloat [▶ 163]	Fügt einen neuen Knoten unterhalb eines existierenden Knotens ein (Float).
AppendChildAsInt [▶ 164]	Fügt einen neuen Knoten unterhalb eines existierenden Knotens ein (Integer).
AppendChildAsLint [▶ 165]	Fügt einen neuen Knoten unterhalb eines existierenden Knotens ein (Integer64).
AppendChildAsUint [▶ 166]	Fügt einen neuen Knoten unterhalb eines existierenden Knotens ein (Unsigned Integer).
AppendChildAsUlint [▶ 166]	Fügt einen neuen Knoten unterhalb eines existierenden Knotens ein (Unsigned Integer64).
AppendCopy [▶ 167]	Fügt einen neuen Knoten unterhalb eines existierenden Knotens ein.
AppendNode [▶ 168]	Fügt einen neuen Knoten unterhalb eines existierenden Knotens ein.
Attribute [▶ 168]	Liest das Attribut eines gegebenen XML-Knotens aus.
AttributeAsBool [▶ 169]	Liefert den Wert eines Attributs als Datentyp Boolean.
AttributeAsDouble [▶ 169]	Liefert den Wert eines Attributs als Datentyp Double.
AttributeAsFloat [▶ 170]	Liefert den Wert eines Attributs als Datentyp Float.
AttributeAsInt [▶ 170]	Liefert den Wert eines Attributs als Datentyp Integer.
AttributeAsLint [▶ 171]	Liefert den Wert eines Attributs als Datentyp Integer64.
AttributeAsUint [▶ 171]	Liefert den Wert eines Attributs als Datentyp Unsigned Integer.
AttributeAsUlint [▶ 172]	Liefert den Wert eines Attributs als Datentyp Unsigned Integer64.
AttributeBegin [▶ 172]	Liefert einen Iterator über alle Attribute eines XML-Knotens.
AttributeFromIterator [▶ 173]	Konvertiert die aktuelle Position eines Iterators in ein XML-Attribut-Objekt.
AttributeName [▶ 174]	Liefert den Namen eines gegebenen Attributs.
Attributes [▶ 174]	Dient zur Navigation durch den DOM und liefert einen Iterator auf alle gefundenen Attribute an einem XML-Knoten zurück.
AttributeText [▶ 175]	Liefert den Text eines gegebenen Attributs.
Begin [▶ 175]	Liefert einen Iterator über alle Kindelemente eines XML-Knotens.
BeginByName [▶ 176]	Liefert einen Iterator über alle Kindelemente eines XML-Knotens, beginnend ab einem bestimmten Element.
Child [▶ 177]	Dient zur Navigation durch den DOM und liefert eine Referenz auf das (erste) Kindelement des aktuellen Knotens zurück.

Name	Beschreibung
ChildByAttribute [▶ 177]	Dient zur Navigation durch den DOM und liefert eine Referenz auf ein Kindelement im XML-Dokument zurück.
ChildByAttributeAndName [▶ 178]	Dient zur Navigation durch den DOM und liefert eine Referenz auf ein Kindelement im XML-Dokument zurück.
ChildByName [▶ 179]	Dient zur Navigation durch den DOM und liefert eine Referenz auf ein Kindelement im XML-Dokument zurück.
Children [▶ 179]	Dient zur Navigation durch den DOM und liefert einen Iterator auf mehrere gefundene Kindelemente im XML-Dokument zurück.
ChildrenByName [▶ 180]	Dient zur Navigation durch den DOM und liefert einen Iterator auf mehrere gefundene Kindelemente im XML-Dokument zurück.
ClearIterator [▶ 181]	Diese Methode setzt bereits verwendete Iteratoren zurück, sodass diese bei dem nächsten Programmdurchlauf wieder verwendet werden können.
Compare [▶ 181]	Überprüft zwei Iteratoren auf Gleichheit.
CopyAttributeText [▶ 182]	Liest den Wert eines XML-Attributs aus und schreibt diesen in eine Variable vom Datentyp String.
CopyDocument [▶ 182]	Kopiert den Inhalt des DOM-Speichers in eine Variable vom Datentyp String.
CopyNodeText [▶ 183]	Liest den Wert eines XML-Knotens aus und schreibt diesen in eine Variable vom Datentyp String.
CopyNodeXml [▶ 184]	Liest die XML-Struktur eines XML-Knotens aus und schreibt diese in eine Variable vom Datentyp String.
End	
FirstNodeByPath [▶ 185]	Navigiert anhand eines an die Methode übergebenen Pfads durch ein XML-Dokument.
GetAttributeTextLength [▶ 185]	Liefert die Länge des Werts eines XML-Attributs.
GetDocumentLength [▶ 186]	Liefert die Länge eines XML-Dokuments in Bytes.
GetDocumentNode [▶ 186]	Liefert den Root-Knoten eines XML-Dokuments.
GetNodeTextLength [▶ 186]	Liefert die Länge des Werts eines XML-Knotens.
GetNodeXmlLength [▶ 187]	Liefert die Länge der XML-Struktur eines XML-Knotens.
GetRootNode [▶ 187]	Liefert eine Referenz zum ersten XML-Knoten im XML-Dokument.
InsertAttributeCopy [▶ 188]	Fügt ein Attribut zu einem XML-Knoten hinzu und kopiert hierbei den Namen und den Wert eines existierenden Attributs.
InsertAttribute [▶ 188]	Fügt ein Attribut zu einem XML-Knoten hinzu.
InsertChild [▶ 189]	Fügt einen Knoten zu einem existierenden XML-Knoten hinzu.
InsertCopy [▶ 190]	Fügt einen neuen Knoten zu einem existierenden XML Knoten hinzu und kopiert hierbei einen existierenden Knoten.
IsEnd [▶ 191]	Überprüft, ob sich ein gegebener XML-Iterator am Ende der zu durchlaufenden Iteration befindet.
LoadDocumentFromFile [▶ 191]	Lädt ein XML-Dokument aus einer Datei.
NewDocument [▶ 192]	Erzeugt ein leeres XML-Dokument im DOM-Speicher.
Next [▶ 192]	Setzt einen XML-Iterator auf das nächste zu durchlaufende Objekt.
NextAttribute [▶ 193]	Liefert zu einem gegebenen XML-Attribut das nächste Attribut.
NextByName [▶ 193]	Setzt einen XML-Iterator auf das nächste zu durchlaufende Objekt, das anhand seines Namens identifiziert wird.
NextSibling [▶ 194]	Liefert zu einem gegebenen XML-Knoten den nächsten direkten Knoten auf derselben XML-Ebene.
NextSiblingByName [▶ 195]	Liefert zu einem gegebenen XML-Knoten den nächsten direkten Knoten mit einem bestimmten Namen auf derselben XML-Ebene.
Node [▶ 195]	Wird in Zusammenhang mit einem Iterator verwendet, um durch den DOM zu navigieren.

Name	Beschreibung
NodeAsBool [▶ 196]	Liefert den Text eines XML-Knotens als Datentyp Boolean.
NodeAsDouble [▶ 196]	Liefert den Text eines XML-Knotens als Datentyp Double.
NodeAsFloat [▶ 197]	Liefert den Text eines XML-Knotens als Datentyp Float.
NodeAsInt [▶ 197]	Liefert den Text eines XML-Knotens als Datentyp Integer.
NodeAsLint [▶ 198]	Liefert den Text eines XML-Knotens als Datentyp Integer64.
NodeAsUint [▶ 198]	Liefert den Text eines XML-Knotens als Datentyp Unsigned Integer.
NodeAsUlint [▶ 199]	Liefert den Text eines XML-Knotens als Datentyp Unsigned Integer64.
NodeName [▶ 199]	Gibt den Namen eines XML-Knotens zurück.
NodeText [▶ 200]	Liefert den Text eines XML-Knotens.
ParseDocument [▶ 200]	Lädt ein XML-Dokument zur weiteren Verarbeitung in den DOM-Speicher.
RemoveAttribute	
RemoveAttributeByName	
RemoveChild [▶ 201]	entfernt einen XML-Kindknoten von einem gegebenen XML-Knoten.
RemoveChildByName [▶ 202]	Entfernt einen XML-Kindknoten von einem gegebenen XML-Knoten.
SaveDocumentToFile [▶ 202]	Speichert das aktuelle XML-Dokument in einer Datei.
SetAttribute [▶ 203]	Setzt den Wert eines Attributs.
SetAttributeAsBool [▶ 204]	Setzt den Wert eines Attributs (Boolean).
SetAttributeAsDouble [▶ 205]	Setzt den Wert eines Attributs (Double).
SetAttributeAsFloat [▶ 205]	Setzt den Wert eines Attributs (Float).
SetAttributeAsInt [▶ 206]	Setzt den Wert eines Attributs (Integer).
SetAttributeAsLint [▶ 206]	Setzt den Wert eines Attributs (Integer64).
SetAttributeAsUint [▶ 207]	Setzt den Wert eines Attributs (Unsigned Integer).
SetAttributeAsUlint [▶ 207]	Setzt den Wert eines Attributs (Unsigned Integer64).
SetChild [▶ 208]	Setzt den Wert eines XML-Knotens (String).
SetChildAsBool [▶ 208]	Setzt den Wert eines XML-Knotens (Boolean).
SetChildAsDouble [▶ 209]	Setzt den Wert eines XML-Knotens (Double).
SetChildAsFloat [▶ 209]	Setzt den Wert eines XML-Knotens (Float).
SetChildAsInt [▶ 210]	Setzt den Wert eines XML-Knotens (Integer).
SetChildAsLint [▶ 211]	Setzt den Wert eines XML-Knotens (Integer64).
SetChildAsUint [▶ 211]	Setzt den Wert eines XML-Knotens (Unsigned Integer).
SetChildAsUlint [▶ 212]	Setzt den Wert eines XML-Knotens (Unsigned Integer64).

Voraussetzungen

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

Sehen Sie dazu auch

- AppendAttribute [▶ 155]
- AppendAttributeAsBool [▶ 155]
- AppendAttributeAsDouble [▶ 156]
- AppendAttributeAsFloat [▶ 157]
- AppendAttributeAsInt [▶ 157]
- AppendAttributeAsLint [▶ 158]
- AppendAttributeAsUint [▶ 159]

4.7.1 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');
```

4.7.2 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);
```

4.7.3 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);
```

4.7.4 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);
```

4.7.5 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);
```

4.7.6 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);
```

4.7.7 AppendAttributeAsUint

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

Syntax

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

Rückgabewert

Name	Typ
AppendAttributeAsUint	SXmlAttribute

Eingänge

Name	Typ
n	SXmlNode
value	UDINT

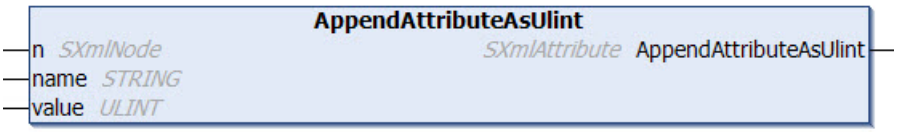
/ Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

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

4.7.8 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);
```

4.7.9 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);
```

4.7.10 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);
```

4.7.11 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

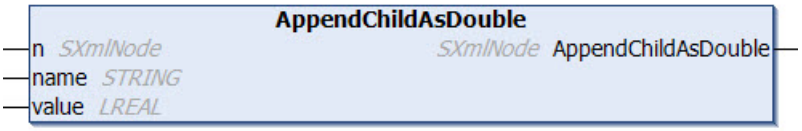
/ Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

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

4.7.12 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);
```

4.7.13 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

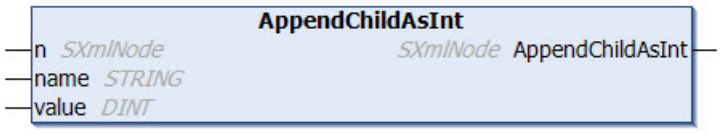
/ Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

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

4.7.14 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);
```

4.7.15 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

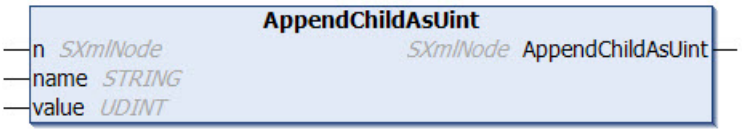
/ Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

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

4.7.16 AppendChildAsUint



Diese Methode fügt einen neuen Knoten unterhalb eines existierenden Knotens ein. Der Wert des neuen Knotens ist hierbei vom Datentyp Unsigned Integer. Der Name und der Wert des neuen Knotens sowie eine Referenz auf den existierenden Knoten werden der Methode als Eingangsparameter übergeben. Die Methode liefert eine Referenz auf den neu hinzugefügten Knoten.

Syntax

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

Rückgabewert

Name	Typ
AppendChildAsUint	SXmlNode

Eingänge

Name	Typ
n	SXmlNode
value	UDINT

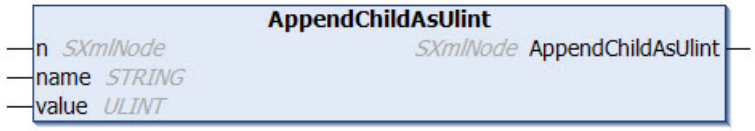
/ Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

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

4.7.17 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

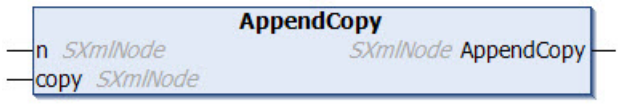
/ Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

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

4.7.18 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

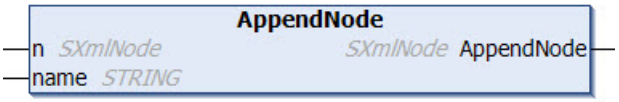
Eingänge

Name	Typ
n	SXmlNode
copy	SXmlNode

Beispielaufruf:

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

4.7.19 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');
```

4.7.20 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

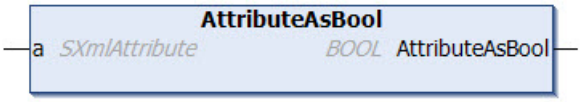
/ Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

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

4.7.21 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

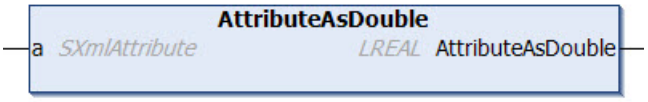
Eingänge

Name	Typ
a	SXmlAttribute

Beispielaufruf:

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

4.7.22 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

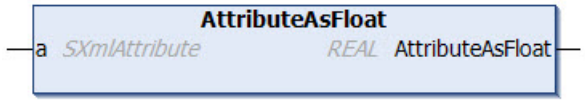
Eingänge

Name	Typ
a	SXmlAttribute

Beispielaufruf:

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

4.7.23 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);
```

4.7.24 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);
```

4.7.25 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);
```

4.7.26 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);
```

4.7.27 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);
```

4.7.28 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

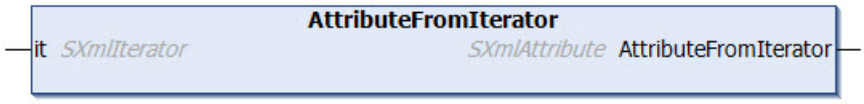
Eingänge

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
```

4.7.29 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

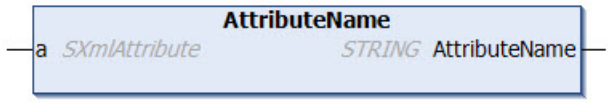
Eingänge

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
```

4.7.30 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);
```

4.7.31 Attributes

Diese Methode dient zur Navigation durch den DOM und liefert einen Iterator auf alle gefundenen Attribute an einem XML-Knoten zurück. Der Iterator kann anschließend zur weiteren Navigation durch die gefundenen Elemente verwendet werden. Der Knoten und eine Referenz auf den Iterator werden der Methode als Eingangsparameter übergeben.

Syntax

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

Rückgabewert

Name	Typ
Attributes	SXmlAttribute

Eingänge

Name	Typ
it	REFERENCE TO SXmlIterator

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
```

4.7.32 AttributeText

Diese Methode liefert den Text eines gegebenen Attributs. Das Attribut wird der Methode als Eingangsparameter übergeben.

Syntax

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

Rückgabewert

Name	Typ
AttributeText	STRING(255)

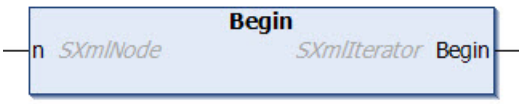
Eingänge

Name	Typ
a	SXmlAttribute

Beispielaufruf:

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

4.7.33 Begin



Diese Methode liefert einen Iterator über alle Kindelemente eines XML-Knotens, immer beginnend ab dem ersten Kindelement. Der XML-Knoten wird der Methode als Eingangsparameter übergeben.

Syntax

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

Rückgabewert

Name	Typ
Begin	SXmlIterator

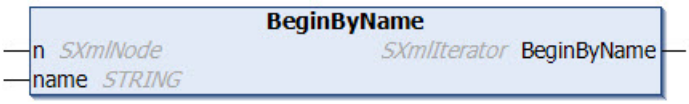
Eingänge

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
```

4.7.34 BeginByName



Diese Methode liefert einen Iterator über alle Kindelemente eines XML-Knotens, beginnend ab einem bestimmten Element. Der XML-Knoten wird der Methode als Eingangsparameter übergeben.

Syntax

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

Rückgabewert

Name	Typ
BeginByName	SXmlIterator

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
```

4.7.35 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);
```

4.7.36 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');
```

4.7.37 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

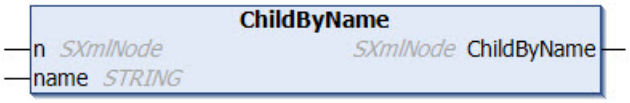
/ Ein-/Ausgänge

Name	Typ
attr	STRING
value	STRING
child	STRING

Beispielaufruf:

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

4.7.38 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.ChildByName(xmlDoc, 'Machines');
```

4.7.39 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

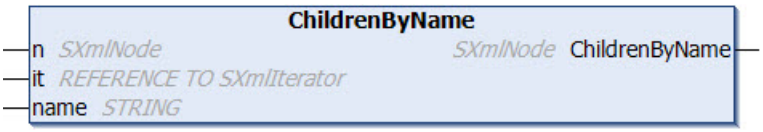
Eingänge

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
```

4.7.40 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

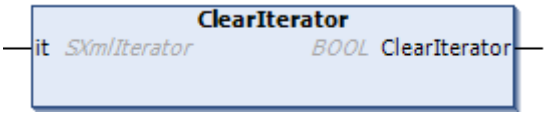
/ Ein-/Ausgänge

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
```

4.7.41 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

4.7.42 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);
```

4.7.43 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

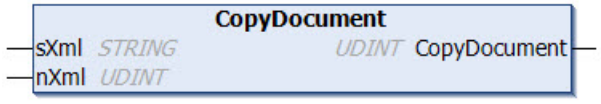
/ Ein-/Ausgänge

Name	Typ
sXml	STRING

Beispielaufruf:

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

4.7.44 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

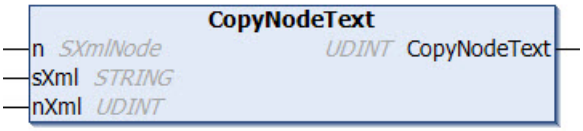
/ Ein-/Ausgänge

Name	Typ
sXml	STRING

Beispielaufruf:

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

4.7.45 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));
```

4.7.46 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

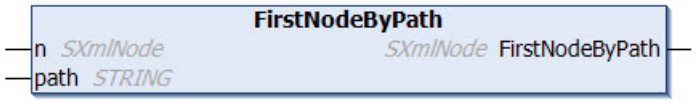
/ Ein-/Ausgänge

Name	Typ
sXml	STRING

Beispielaufruf:

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

4.7.47 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');
```

4.7.48 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);
```

4.7.49 GetDocumentLength

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

Syntax

```
METHOD GetDocumentLength : UDINT
```

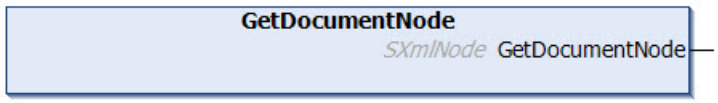
Rückgabewert

Name	Typ
GetDocumentLength	UDINT

Beispielaufruf:

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

4.7.50 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();
```

4.7.51 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

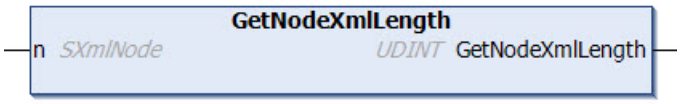
Eingänge

Name	Typ
n	SXmlNode

Beispielaufruf:

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

4.7.52 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

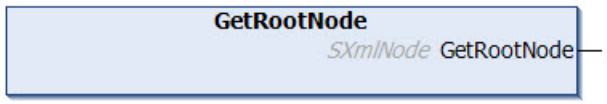
Eingänge

Name	Typ
n	SXmlNode

Beispielaufruf:

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

4.7.53 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();
```

4.7.54 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);
```

4.7.55 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');
```

4.7.56 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');
```

4.7.57 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);
```

4.7.58 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

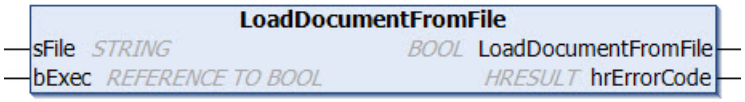
Eingänge

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
```

4.7.59 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.

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

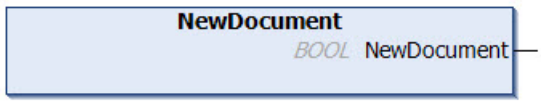
Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

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

4.7.60 NewDocument



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

Syntax

```
METHOD NewDocument : BOOL
```

Rückgabewert

Name	Typ
NewDocument	BOOL

Beispielaufruf:

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

4.7.61 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
```

4.7.62 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

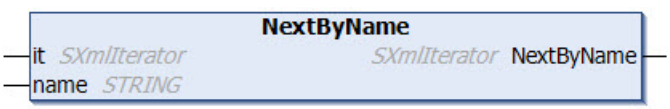
Eingänge

Name	Typ
a	SXmlAttribute

Beispielaufruf:

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

4.7.63 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
```

4.7.64 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);
```

4.7.65 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

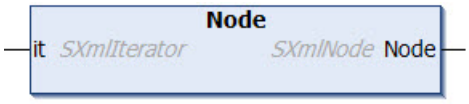
/ Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

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

4.7.66 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

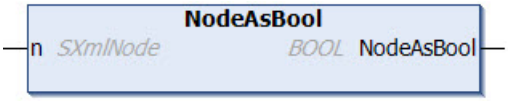
Eingänge

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
```

4.7.67 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);
```

4.7.68 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

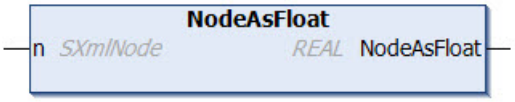
Eingänge

Name	Typ
n	SXmlNode

Beispielaufruf:

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

4.7.69 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);
```

4.7.70 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);
```

4.7.71 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

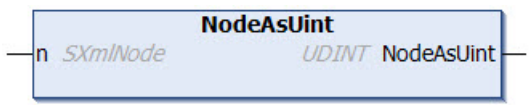
Eingänge

Name	Typ
n	SXmlNode

Beispielaufruf:

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

4.7.72 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);
```

4.7.73 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

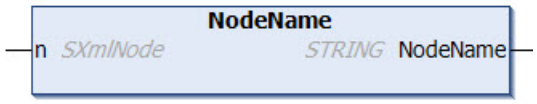
Eingänge

Name	Typ
n	SXmlNode

Beispielaufruf:

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

4.7.74 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

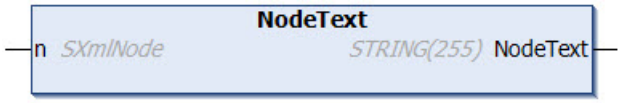
Eingänge

Name	Typ
n	SXmlNode

Beispielaufruf:

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

4.7.75 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 [▶ 183] verwendet werden.

Syntax

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

Rückgabewert

Name	Typ
NodeText	STRING(255)

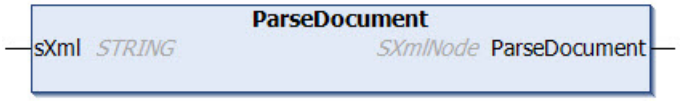
Eingänge

Name	Typ
n	SXmlNode

Beispielaufruf:

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

4.7.76 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

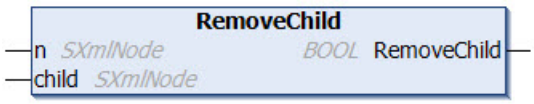
/ Ein-/Ausgänge

Name	Typ
sXml	STRING

Beispielaufruf:

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

4.7.77 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);
```

4.7.78 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

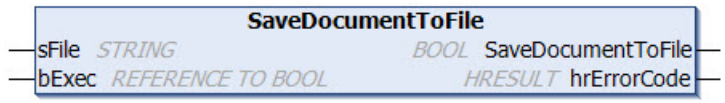
/ Ein-/Ausgänge

Name	Typ
name	STRING

Beispielaufruf:

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

4.7.79 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

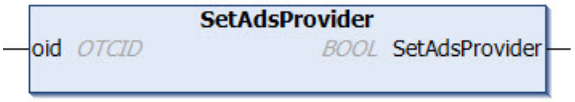
Ausgänge

Name	Typ
hrErrorCode	HRESULT

Beispielaufruf:

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

4.7.80 SetAdsProvider



Syntax

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

Rückgabewert

Name	Typ
SetAdsProvider	BOOL

/ Ein-/Ausgänge

Name	Typ
oid	OTCID

4.7.81 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');
```

4.7.82 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

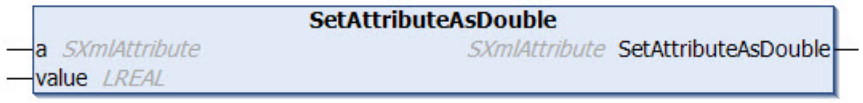
Eingänge

Name	Typ
a	SXmlAttribute
value	BOOL

Beispielaufruf:

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

4.7.83 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

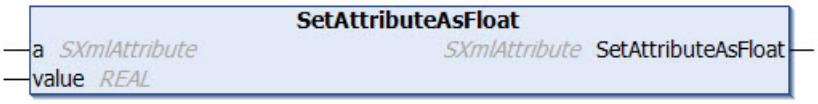
Eingänge

Name	Typ
a	SXmlAttribute
value	LREAL

Beispielaufruf:

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

4.7.84 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

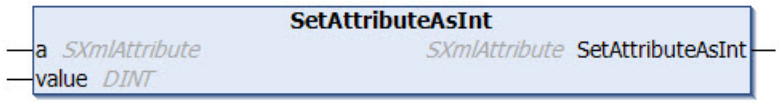
Eingänge

Name	Typ
a	SXmlAttribute
value	REAL

Beispielaufruf:

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

4.7.85 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);
```

4.7.86 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);
```

4.7.87 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

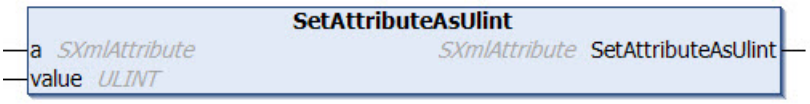
Eingänge

Name	Typ
a	SXmlAttribute
value	UDINT

Beispielaufruf:

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

4.7.88 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);
```

4.7.89 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);
```

4.7.90 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);
```

4.7.91 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);
```

4.7.92 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);
```

4.7.93 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);
```

4.7.94 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);
```

4.7.95 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);
```

4.7.96 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

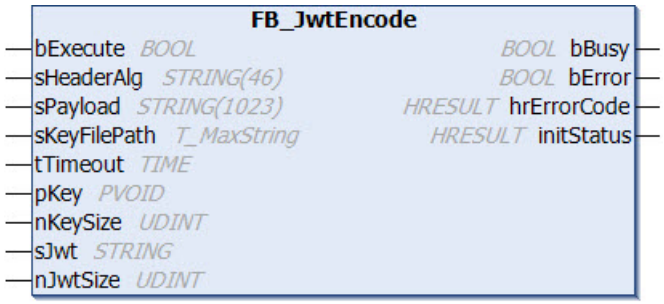
Eingänge

Name	Typ
n	SXmlNode
value	ULINT

Beispielaufruf:

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

4.8 FB_JwtEncode



Der Funktionsbaustein ermöglicht die Erzeugung und Signierung eines JSON Web Token (JWT).

Syntax

Definition:

```
FUNCTION_BLOCK FB_JwtEncode
VAR_INPUT
  bExecute       : BOOL;
  sHeaderAlg     : STRING(46);
  sPayload       : STRING(1023);
  sKeyFilePath   : STRING(511);
  tTimeout       : TIME;
  pKey           : PVOID;
  nKeySize       : UDINT;
```

```
  nJwtSize       : UDINT;
END_VAR
VAR_IN_OUT CONSTANT
  sJwt           : STRING;
END_VAR
VAR_OUTPUT
  bBusy          : BOOL;
  bError         : BOOL;
  hrErrorCode    : HRESULT;
  initStatus     : HRESULT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
bExecute	BOOL	Steigende Flanke aktiviert die Abarbeitung des Funktionsbausteins.
sHeaderAlg	STRING(46)	Der zu verwendende Algorithmus für den JWT Header, z. B. RS256.
sPayload	STRING(1023	Der zu verwendende Payload des JWT.
sKeyFilePath	STRING(511)	Pfad zum Private Key, welcher für die Signatur des JWT verwendet werden soll.
tTimeout	TIME	ADS Timeout, welcher intern für den Dateizugriff auf den Private Key verwendet wird.
pKey	PVOID	Buffer für den auszulesenden Private Key.
nKeySize	UDINT	Maximalgröße des Buffers.
sJwt	STRING	Enthält nach Abarbeitung des Funktionsbausteins das fertig kodierte und signierte JWT.
nJwtSize	UDINT	Größe des erzeugten JWT inklusive Nullterminierung.

Ausgänge

Name	Typ	Beschreibung
bBusy	BOOL	Ist TRUE, solange die Abarbeitung des Funktionsbausteins noch nicht beendet ist.
bError	BOOL	Wird TRUE, sobald eine Fehlersituation eintritt.
hrErrorCode	HRESULT	Liefert bei einem gesetzten Ausgang `bError` einen Fehlercode. Eine Erläuterung zu den möglichen Fehlercodes befindet sich im Anhang.
initStatus	HRESULT	Liefert im Fall einer fehlgeschlagenen Initialisierung des Funktionsbausteins einen Fehlercode.

Voraussetzungen

TwinCAT Version	Hardware	Einzubindende Bibliotheken
TwinCAT 3.1, Build 4024.4	x86, x64, ARM	Tc3_JsonXml 3.3.6.0

5 Schnittstellen

5.1 ITcJsonSaxHandler

5.1.1 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
```

5.1.2 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
```

5.1.3 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
```

5.1.4 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
```

5.1.5 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
```

5.1.6 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
```

5.1.7 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
```

5.1.8 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
```

5.1.9 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
```

5.1.10 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
```

5.1.11 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
```

5.1.12 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
```

5.1.13 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 ITcJsonSaxValues

5.2.1 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 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
```

5.2.3 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
```

5.2.4 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
```

5.2.5 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
```

5.2.6 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
```

5.2.7 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
```

5.2.8 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.

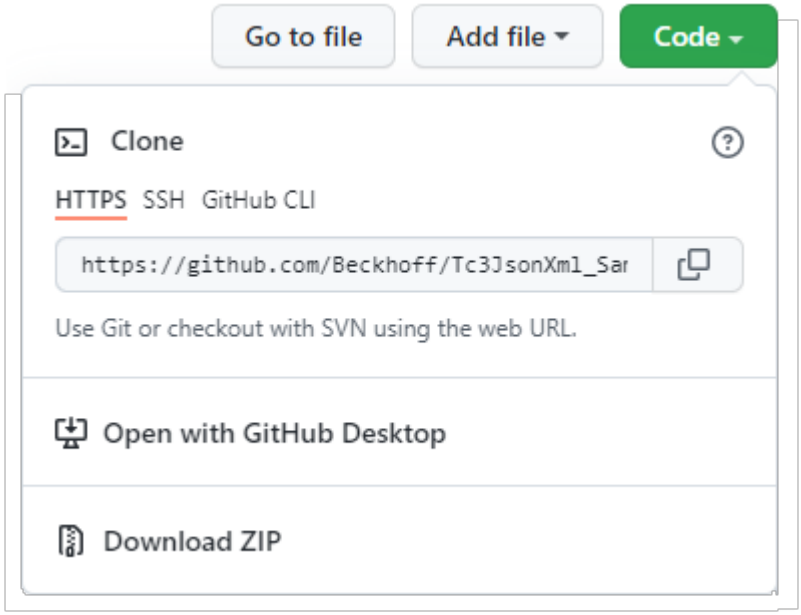
Syntax

```
METHOD OnUlintValue : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
    value : ULINT;
END_VAR
```

6 Beispiele

Downloads

Beispielcode und -konfigurationen für dieses Produkt können über das entsprechende Repository auf GitHub bezogen werden: https://github.com/Beckhoff/Tc3JsonXml_Samples. Sie haben dort die Möglichkeit das Repository zu clonen oder ein ZIP File mit dem Sample herunterzuladen.



6.1 Tc3JsonXmlSampleJsonDataType

Beispiel zum automatischen Konvertieren von Strukturen in eine JSON-Nachricht

Dieses Beispiel veranschaulicht, wie eine Datenstruktur in eine JSON-Nachricht (und umgekehrt) konvertiert werden kann. Bei der Konvertierung wird der Aufbau einer Struktur eins-zu-eins in ein entsprechendes JSON-Äquivalent überführt. Über SPS-Attribute an den Member-Variablen der Struktur können zusätzlich Metadaten angelegt werden.

Aufbau der zu konvertierenden Datenstruktur

```
TYPE ST_Values :
STRUCT

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

  {attribute 'Unit' := 'V'}
  {attribute 'DisplayName' := 'Voltage'}
  Sensor2 : DINT;

  {attribute 'Unit' := 'A'}
  {attribute 'DisplayName' := 'Current'}
  Sensor3 : DINT;

END_STRUCT
END_TYPE
```

Deklarationsbereich

```
PROGRAM MAIN
VAR
  dtTimestamp    : DATE_AND_TIME := DT#2017-04-04-12:42:42;
  fbJson         : FB_JsonSaxWriter;
  fbJsonDataType : FB_JsonReadWriteDataType;
  sJsonDoc       : STRING(255);
```

```
  sJsonDoc2      : STRING(2000);
  stValues       : ST_Values;
END_VAR
```

Implementierungsbereich

Ausgehend von der Instanz fbJson des Funktionsbausteins FB_JsonSaxWriter werden zwei Wege zum Generieren der JSON-Nachricht gezeigt. Bei einer JSON-Nachricht mit nicht mehr als 255 Zeichen kann die Methode GetDocument() verwendet werden. Bei größeren JSON-Nachrichten muss hingegen die Methode CopyDocument() verwendet werden.

```
fbJson.ResetDocument();
fbJson.StartObject();
fbjson.AddKeyDateTime('Timestamp', dtTimestamp);
fbJsonDataType.AddJsonKeyValueFromSymbol(fbJson, 'Values', 'ST_Values', SIZEOF(stValues), ADR(stValu
es));
fbJsonDataType.AddJsonKeyPropertiesFromSymbol(fbJson, 'MetaData', 'ST_Values', 'Unit|DisplayName');
fbJson.EndObject();
sJsonDoc := fbJson.GetDocument();
fbJson.CopyDocument(sJsonDoc2, SIZEOF(sJsonDoc2));
```

Resultierende JSON-Nachricht

```
{
  "Timestamp": "2017-04-04T12:42:42",
  "Values": {
    "Sensor1": 0.0,
    "Sensor2": 0,
    "Sensor3": 0
  },
  "MetaData": {
    "Sensor1": {
      "Unit": "m/s",
      "DisplayName": "Speed"
    },
    "Sensor2": {
      "Unit": "V",
      "DisplayName": "Voltage"
    },
    "Sensor3": {
      "Unit": "A",
      "DisplayName": "Current"
    }
  }
}
```

Alternative

Als Alternative kann auch die Methode AddJsonValueFromSymbol() verwendet werden, um aus einer Datenstruktur direkt ein JSON-Format zu erzeugen.

```
fbJson.ResetDocument();
fbJsonDataType.AddJsonValueFromSymbol(fbJson, 'ST_Values', SIZEOF(stValues), ADR(stValues));
sJsonDoc := fbJson.GetDocument();
fbJson.CopyDocument(sJsonDoc2, SIZEOF(sJsonDoc2));
```

Das resultierende JSON-Objekt sieht wie folgt aus:

```
{
  "Sensor1": 0.0,
  "Sensor2": 0,
  "Sensor3": 0
}
```

Konvertierung einer JSON-Nachricht zurück in eine Datenstruktur

Die obigen Beispiele zeigen, wie auf einfache Art und Weise aus einer Datenstruktur ein JSON-Objekt erzeugt werden kann. Für den umgekehrten Weg, also das Extrahieren von Werten aus einem (empfangenen) JSON-Objekt zurück in eine Datenstruktur, gibt es ebenfalls eine entsprechende Methode in der Tc3_JsonXml-Bibliothek. Der Aufruf der Methode SetSymbolFromJson() ermöglicht diesen Anwendungsfall.

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

Die String-Variable sJsonDoc2 enthält das JSON-Objekt, das durch den Aufruf der Methode in die Strukturinstanz stValuesReceive überführt wird.

Zieldatenstruktur

Die Zieldatenstruktur muss zum Aufbau des JSON-Dokuments passen. Ansonsten liefert SetSymbolFromJson() FALSE zurück.

6.2 Tc3JsonXmlSampleJsonSaxReader

Beispiel zum Parsen von JSON-Dokumenten via SAX Reader

Dieses Beispiel veranschaulicht, wie eine JSON-Nachricht programmatisch durchlaufen werden kann. Als Basis wird der Funktionsbaustein FB_JsonSaxReader verwendet.

Deklarationsbereich

```
PROGRAM MAIN
VAR
  fbJson     : FB_JsonSaxReader;
  pJsonParse : JsonSaxHandler;
  sJsonDoc   : STRING(255) := '{"Values":
{"Timestamp":"2017-04-04T12:42:42","Sensor1":42.42,"Sensor2":42}}';
END_VAR
```

Implementierungsbereich

Durch den Aufruf der Methode Parse(), die Übergabe der JSON-Nachricht als STRING und den Interface-Pointer auf eine Funktionsbaustein-Instanz, die das Interface ITcJsonSaxHandler implementiert, werden der SAX Reader aktiviert und die entsprechenden Callback-Methoden durchlaufen.

```
fbJson.Parse(sJson := sJsonDoc, ipHdl := pJsonParse);
```

Callback-Methoden

Die Callback-Methoden werden an der Instanz des Funktionsbausteins, der das Interface ITcJsonSaxHandler implementiert, aufgerufen. Jede Callback-Methode repräsentiert ein „gefundenes“ Element in der JSON-Nachricht. Zum Beispiel wird die Callback-Methode OnStartObject() aufgerufen, sobald eine geöffnete geschweifte Klammer detektiert wurde. Laut der oben genannten Beispiel-JSON-Nachricht werden also die folgenden Callback-Methoden in dieser Reihenfolge durchlaufen:

1. OnStartObject(), aufgrund der ersten geöffneten geschweiften Klammer
2. OnKey(), aufgrund des Properties "Values"
3. OnStartObject(), aufgrund der zweiten geöffneten geschweiften Klammer
4. OnKey(), aufgrund des Properties "Timestamp"
5. OnString(), aufgrund des Werts von Property "Timestamp"
6. OnKey(), aufgrund des Properties "Sensor1"
7. OnLreal(), aufgrund des Werts von Property "Sensor1“"
8. OnKey(), aufgrund des Properties "Sensor2"
9. OnUdint(), aufgrund des Werts von Property "Sensor2"
10. OnEndObject(), aufgrund der ersten geschlossenen geschweiften Klammer
11. OnEndObject(), aufgrund der zweiten geschlossenen geschweiften Klammer

Innerhalb der Callback-Methoden wird der aktuelle Zustand über eine Instanz des Enums E_JsonStates definiert und gespeichert. Hierüber kann auch ermittelt werden, ob es sich um eine gültige JSON-Nachricht handelt. Wenn zum Beispiel die Callback-Methode OnLreal() aufgerufen wird und sich der Zustand nicht im erwarteten State 70 (JSON_STATE_ONLREAL) befindet, kann an die Methode der Rückgabewert S_FALSE zurückgegeben werden. Der SAX Reader beendet dann automatisch die weitere Verarbeitung.

6.3 Tc3JsonXmlSampleJsonSaxWriter

Beispiel zum Erstellen von JSON-Dokumenten via SAX Writer

Dieses Beispiel veranschaulicht, wie eine JSON-Nachricht über den SAX-Mechanismus erstellt werden kann. Als Basis wird der Funktionsbaustein FB_JsonSaxWriter verwendet.

Deklarationsbereich

```
PROGRAM MAIN
VAR
  dtTimestamp : DATE_AND_TIME := DT#2017-04-04-12:42:42;
  fbJson      : FB_JsonSaxWriter;
  sJsonDoc    : STRING(255);
END_VAR
```

Implementierungsbereich

Der SAX-Mechanismus durchläuft ein zu erstellendes JSON-Dokument sequentiell, d. h. die entsprechenden Elemente werden der Reihe nach durchlaufen und erstellt.

```
fbJson.StartObject();
fbJson.AddKey('Timestamp');
fbJson.AddDateTime(dtTimestamp);
fbJson.AddKey('Values');
fbJson.StartObject();
fbJson.AddKey('Sensor1');
fbJson.AddReal(42.42);
fbJson.AddKey('Sensor2');
fbJson.AddDint(42);
fbJson.AddKey('Sensor3');
fbJson.AddBool(TRUE);
fbJson.EndObject();
fbJson.EndObject();
sJsonDoc := fbJson.GetDocument();
fbJson.ResetDocument();
```

Resultierende JSON-Nachricht

```
{
  "Timestamp": "2017-04-04T12:42:42",
  "Values": {
    "Sensor1": 42.42,
    "Sensor2": 42,
    "Sensor3": true
  }
}
```

6.4 Tc3JsonXmlSampleJsonDomReader

Dieses Beispiel veranschaulicht, wie eine JSON-Nachricht programmatisch auf Basis von DOM durchlaufen werden kann. Als Basis wird der Funktionsbaustein FB_JsonDomParser verwendet.

Deklarationsbereich

```
PROGRAM MAIN
VAR
  fbJson     : FB_JsonDomParser;
  jsonDoc    : SJsonValue;
  jsonProp   : SJsonValue;
  jsonValue  : SJsonValue;
  bHasMember : BOOL;
  sMessage   : STRING(255) := '{"serialNumber":"G030PT028191AC4R","batteryVoltage":"1547mV","clickTy
pe":"SINGLE"}';
  stReceivedData : ST_ReceivedData;
END_VAR
```

Implementierungsbereich

Durch die Methode ParseDocument() wird die JSON-Nachricht in den DOM-Tree geladen. Anschließend kann mit der Methode HasMember() überprüft werden, ob ein bestimmtes Property enthalten ist. Über die Methode FindMember() wird das Property selektiert und über GetString() dessen Value extrahiert.

```
jsonDoc := fbJson.ParseDocument(sMessage);
bHasMember := fbJson.HasMember(jsonDoc, 'serialNumber');
IF (bHasMember) THEN
  bHasMember := FALSE;
  jsonProp := fbJson.FindMember(jsonDoc, 'serialNumber');
```

```
  stReceivedData.serialNumber := fbJson.GetString(jsonProp);
END_IF

bHasMember := fbJson.HasMember(jsonDoc, 'batteryVoltage');
IF (bHasMember) THEN
  bHasMember := FALSE;
  jsonProp := fbJson.FindMember(jsonDoc, 'batteryVoltage');
  stReceivedData.batteryVoltage := fbJson.GetString(jsonProp);
END_IF

bHasMember := fbJson.HasMember(jsonDoc, 'clickType');
IF (bHasMember) THEN
  bHasMember := FALSE;
  jsonProp := fbJson.FindMember(jsonDoc, 'clickType');
  stReceivedData.clickType := fbJson.GetString(jsonProp);
END_IF
```

Die Verwendung der Methode HasMember() ist nicht zwingend erforderlich, da die Methode FindMember() bereits 0 zurückliefert, wenn ein Property nicht gefunden wurde. Der oben dargestellte Code kann also auch wie folgt implementiert werden:

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

jsonProp := fbJson.FindMember(jsonDoc, 'serialNumber');
IF (jsonProp <> 0) THEN
  stReceivedData.serialNumber := fbJson.GetString(jsonProp);
END_IF

jsonProp := fbJson.FindMember(jsonDoc, 'batteryVoltage');
IF (jsonProp <> 0) THEN
  stReceivedData.batteryVoltage := fbJson.GetString(jsonProp);
END_IF

jsonProp := fbJson.FindMember(jsonDoc, 'clickType');
IF (jsonProp <> 0) THEN
  stReceivedData.clickType := fbJson.GetString(jsonProp);
END_IF
```

Verschachtelte JSON-Objekte

Bei verschachtelten JSON-Objekten ist die Herangehensweise ähnlich. Dadurch, dass sich das gesamte Dokument im DOM befindet, kann darin einfach navigiert werden. Gegeben sei ein JSON-Objekt, das sich wie folgt darstellt:

```
sMessage : STRING(255) := '{"Values":{"serial":"G030PT028191AC4R"}}';
```

Das gesuchte Property befindet sich in dem Unterobjekt „Values“. Der folgende Code zeigt, wie das Property extrahiert werden kann.

```
jsonDoc := fbJson.ParseDocument(sMessage);
bHasMember := fbJson.HasMember(jsonDoc, 'Values');
IF (bHasMember) THEN
  bHasMember := FALSE;
  jsonProp := fbJson.FindMember(jsonDoc, 'Values');
  IF jsonProp <> 0 THEN
    jsonSerial := fbJson.FindMember(jsonProp, 'serial');
    stReceivedData.serialNumber := fbJson.GetString(jsonSerial);
  END_IF
END_IF
```

6.5 Tc3JsonXmlSampleXmlDomReader

Dieses Beispiel veranschaulicht, wie ein XML-Dokument programmatisch auf Basis von DOM durchlaufen werden kann. Als Basis wird der Funktionsbaustein FB_XmlDomParser verwendet.

Deklarationsbereich

```
PROGRAM MAIN
VAR
  fbXml : FB_XmlDomParser;
  xmlDoc : SXmlNode;
  xmlMachines : SXmlNode;
  xmlMachine1 : SXmlNode;
  xmlMachine2 : SXmlNode;
  xmlIterator : SXmlIterator;
```

```
  xmlMachineNode : SXmlNode;
  xmlMachineNodeValue : STRING;
  xmlMachineAttributeRef : SXmlAttribute;
  xmlMachine1Attribute : SXmlAttribute;
  xmlMachine2Attribute : SXmlAttribute;
  sMachine1Name : STRING;
  sMachine2Name : STRING;
  nMachineAttribute : DINT;
  nMachine1Attribute : DINT;
  nMachine2Attribute : DINT;
  sMessageToParse : STRING(255) := '<Machines><Machine Type="1" Test="3">Wilde Nelli</
Machine><Machine Type="2">Huber8</Machine></Machines>';
END_VAR
```

Implementierungsbereich

Der Implementierungsbereich zeigt verschiedene Möglichkeiten auf, wie man ein XML-Dokument parsen kann.

```
(* Load XML content *)
xmlDoc := fbXml.ParseDocument(sMessageToParse);

(* Parse XML nodes - Option 1 *)
xmlMachines := fbXml.ChildByName(xmlDoc, 'Machines');
xmlMachine1 := fbXml.ChildByAttribute(xmlMachines, 'Type', '1');
xmlMachine2 := fbXml.ChildByAttributeAndName(xmlMachines, 'Type', '2', 'Machine');

(* Parse XML nodes - Option 2 *)
xmlMachines := fbXml.ChildByName(xmlDoc, 'Machines');
xmlMachineNode := fbXml.Children(xmlMachines, xmlIterator);
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlMachineNode := fbXml.Node(xmlIterator);
  xmlMachineNodeValue := fbXml.NodeText(xmlMachineNode);
  xmlMachineNode := fbXml.Children(xmlMachines, xmlIterator);
END_WHILE

(* Parse XML nodes - Option 3 *)
xmlMachines := fbXml.ChildByName(xmlDoc, 'Machines');
xmlIterator := fbXml.Begin(xmlMachines);
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlMachineNode := fbXml.Node(xmlIterator);
  xmlMachineNodeValue := fbXml.NodeText(xmlMachineNode);
  xmlIterator := fbXml.Next(xmlIterator);
  xmlIterator := fbXml.End(xmlMachines);
END_WHILE

(* Parse XML attributes - Option 1*)
xmlMachine1Attribute := fbXml.Attribute(xmlMachine1, 'Type');
xmlMachine2Attribute := fbXml.Attribute(xmlMachine2, 'Type');

(* Parse XML attributes - Option 2*)
xmlIterator := fbXml.AttributeBegin(xmlMachine1);
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlMachineAttributeRef := fbXml.AttributeFromIterator(xmlIterator);
  nMachineAttribute := fbXml.AttributeAsInt(xmlMachineAttributeRef);
  xmlIterator := fbXml.Next(xmlIterator);
END_WHILE

(* Retrieve node values *)
sMachine1Name := fbXml.NodeText(xmlMachine1);
sMachine2Name := fbXml.NodeText(xmlMachine2);

(* Retrieve attribute values *)
nMachine1Attribute := fbXml.AttributeAsInt(xmlMachine1Attribute);
nMachine2Attribute := fbXml.AttributeAsInt(xmlMachine2Attribute);
```

6.6 Tc3JsonXmlSampleXmlDomWriter

Dieses Beispiel veranschaulicht, wie ein XML-Dokument programmatisch auf Basis von DOM erstellt werden kann. Als Basis wird der Funktionsbaustein FB_XmlDomParser verwendet.

Deklarationsbereich

```
PROGRAM MAIN
VAR
  fbXml : FB_XmlDomParser;
```

```
  objRoot : SXmlNode;
  objMachines : SXmlNode;
  objMachine : SXmlNode;
  objControllers : SXmlNode;
  objController : SXmlNode;
  objAttribute : SXmlAttribute;
  sXmlString : STRING(1000);
  bCreate : BOOL := FALSE;
  bSave : BOOL := TRUE;
  nLength : UDINT;
  newAttr : SXmlAttribute;
END_VAR
```

Implementierungsbereich

Der Implementierungsbereich zeigt verschiedene Möglichkeiten auf, wie ein XML-Dokument erstellt werden kann.

```
IF bCreate THEN
  (* Create an empty XML document *)
  objRoot := fbXml.GetDocumentNode();

  (* Create a new XML node 'Machines' and add to the empty document *)
  objMachines := fbXml.AppendNode(objRoot, 'Machines');

  (* Create a new XML node 'Machine' and add an attribute to this node. Append node to 'Machines' *)
  objMachine := fbXml.AppendNode(objMachines, 'Machine');
  objAttribute := fbXml.AppendAttribute(objMachine, 'Name', 'Wilde Nelli');

  (* Create a new XML node 'Controllers' and add to the 'Machine' node *)
  objControllers := fbXml.AppendNode(objMachine, 'Controllers');

  (* Create a new XML node 'Controller' and add some attributes. Append node to 'Controllers'. *)
  objController := fbXml.AppendChild(objControllers, 'Controller', 'CX5120', FALSE);
  objAttribute := fbXml.AppendAttribute(objController, 'Type', 'EPC');
  objAttribute := fbXml.AppendAttribute(objController, 'OS', 'Windows Embedded Compact 7');

  (* Create a new XML node 'Controller' and add some attributes. Append node to 'Controllers'. *)
  objController := fbXml.AppendChild(objControllers, 'Controller', 'CX2040', FALSE);
  objAttribute := fbXml.AppendAttribute(objController, 'Type', 'EPC');
  objAttribute := fbXml.AppendAttribute(objController, 'OS', 'Windows Embedded Standard 7');

  (* Create a new XML node 'Controller' and add some attributes. Append node to 'Controllers'. *)
  objController := fbXml.AppendChild(objControllers, 'Controller', 'C6015', FALSE);
  objAttribute := fbXml.AppendAttribute(objController, 'Type', 'IPC');
  objAttribute := fbXml.AppendAttribute(objController, 'OS', 'Windows 10 IoT Enterprise');

  (* Create a new XML node 'Machine' and add an attribute to this node. Append node to 'Machines' *)
  objMachine := fbXml.AppendNode(objMachines, 'Machine');
  objAttribute := fbXml.AppendAttribute(objMachine, 'Name', 'Stanze Oscar');

  (* Create a new XML node 'Controllers' and add to the 'Machine' node *)
  objControllers := fbXml.AppendNode(objMachine, 'Controllers');

  (* Create a new XML node 'Controller' and add some attributes. Append node to 'Controllers'. *)
  objController := fbXml.AppendChild(objControllers, 'Controller', 'C6017', FALSE);
  objAttribute := fbXml.AppendAttribute(objController, 'Type', 'IPC');
  objAttribute := fbXml.AppendAttribute(objController, 'OS', 'Windows 10 IoT Enterprise');
  newAttr := fbXml.InsertAttribute(objController, objAttribute, 'AddAttribute');
  fbXml.SetAttribute(newAttr, 'Hola');

  (* Retrieve XML document and store in a variable of data type STRING(1000) *)
  nLength := fbXml.CopyDocument(sXmlString, SIZEOF(sXmlString));
  bCreate := FALSE;
END_IF
```

7 Fehlercodes

7.1 ADS Return Codes

Gruppierung der Fehlercodes:
Globale Fehlercodes: 0x0000 [▶ 226]... (0x9811_0000 ...)
Router Fehlercodes: 0x0500 [▶ 226]... (0x9811_0500 ...)
Allgemeine ADS Fehler: 0x0700 [▶ 227]... (0x9811_0700 ...)
RTime Fehlercodes: 0x1000 [▶ 229]... (0x9811_1000 ...)

Globale Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x0	0	0x98110000	ERR_NOERROR	Kein Fehler.
0x1	1	0x98110001	ERR_INTERNAL	Interner Fehler.
0x2	2	0x98110002	ERR_NORTIME	Keine Echtzeit.
0x3	3	0x98110003	ERR_ALLOCLOCKEDMEM	Zuweisung gesperrt - Speicherfehler.
0x4	4	0x98110004	ERR_INSERTMAILBOX	Postfach voll – Es konnte die ADS Nachricht nicht versendet werden. Reduzieren der Anzahl der ADS Nachrichten pro Zyklus bringt Abhilfe.
0x5	5	0x98110005	ERR_WRONGRECEIVEHMSG	Falsches HMSG.
0x6	6	0x98110006	ERR_TARGETPORTNOTFOUND	Ziel-Port nicht gefunden – ADS Server ist nicht gestartet oder erreichbar.
0x7	7	0x98110007	ERR_TARGETMACHINENOTFOUND	Zielrechner nicht gefunden – AMS Route wurde nicht gefunden.
0x8	8	0x98110008	ERR_UNKNOWNCMDID	Unbekannte Befehl-ID.
0x9	9	0x98110009	ERR_BADTASKID	Ungültige Task-ID.
0xA	10	0x9811000A	ERR_NOIO	Kein IO.
0xB	11	0x9811000B	ERR_UNKNOWNAMSCMD	Unbekannter AMS-Befehl.
0xC	12	0x9811000C	ERR_WIN32ERROR	Win32 Fehler.
0xD	13	0x9811000D	ERR_PORTNOTCONNECTED	Port nicht verbunden.
0xE	14	0x9811000E	ERR_INVALIDAMSLENGTH	Ungültige AMS-Länge.
0xF	15	0x9811000F	ERR_INVALIDAMSNETID	Ungültige AMS Net ID.
0x10	16	0x98110010	ERR_LOWINSTLEVEL	Installations-Level ist zu niedrig –TwinCAT 2 Lizenzfehler.
0x11	17	0x98110011	ERR_NODEBUGINTAVAILABLE	Kein Debugging verfügbar.
0x12	18	0x98110012	ERR_PORTDISABLED	Port deaktiviert – TwinCAT System Service nicht gestartet.
0x13	19	0x98110013	ERR_PORTALREADYCONNECTED	Port bereits verbunden.
0x14	20	0x98110014	ERR_AMSSYNC_W32ERROR	AMS Sync Win32 Fehler.
0x15	21	0x98110015	ERR_AMSSYNC_TIMEOUT	AMS Sync Timeout.
0x16	22	0x98110016	ERR_AMSSYNC_AMSERROR	AMS Sync Fehler.
0x17	23	0x98110017	ERR_AMSSYNC_NOINDEXINMAP	Keine Index-Map für AMS Sync vorhanden.
0x18	24	0x98110018	ERR_INVALIDAMSPORT	Ungültiger AMS-Port.
0x19	25	0x98110019	ERR_NOMEMORY	Kein Speicher.
0x1A	26	0x9811001A	ERR_TCPSEND	TCP Sendefehler.
0x1B	27	0x9811001B	ERR_HOSTUNREACHABLE	Host nicht erreichbar.
0x1C	28	0x9811001C	ERR_INVALIDAMSFRAGMENT	Ungültiges AMS Fragment.
0x1D	29	0x9811001D	ERR_TLSSEND	TLS Sendefehler – Secure ADS Verbindung fehlgeschlagen.
0x1E	30	0x9811001E	ERR_ACCESSDENIED	Zugriff Verweigert – Secure ADS Zugriff verweigert.

Router Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x500	1280	0x98110500	ROUTERERR_NOLOCKEDMEMORY	Lockierter Speicher kann nicht zugewiesen werden.
0x501	1281	0x98110501	ROUTERERR_RESIZEMEMORY	Die Größe des Routerspeichers konnte nicht geändert werden.
0x502	1282	0x98110502	ROUTERERR_MAILBOXFULL	Das Postfach hat die maximale Anzahl der möglichen Meldungen erreicht.
0x503	1283	0x98110503	ROUTERERR_DEBUGBOXFULL	Das Debug Postfach hat die maximale Anzahl der möglichen Meldungen erreicht.
0x504	1284	0x98110504	ROUTERERR_UNKNOWNPORTTYPE	Der Porttyp ist unbekannt.
0x505	1285	0x98110505	ROUTERERR_NOTINITIALIZED	Router ist nicht initialisiert.
0x506	1286	0x98110506	ROUTERERR_PORTALREADYINUSE	Die Portnummer ist bereits vergeben.
0x507	1287	0x98110507	ROUTERERR_NOTREGISTERED	Der Port ist nicht registriert.
0x508	1288	0x98110508	ROUTERERR_NOMOREQUEUES	Die maximale Portanzahl ist erreicht.
0x509	1289	0x98110509	ROUTERERR_INVALIDPORT	Der Port ist ungültig.
0x50A	1290	0x9811050A	ROUTERERR_NOTACTIVATED	Der Router ist nicht aktiv.
0x50B	1291	0x9811050B	ROUTERERR_FRAGMENTBOXFULL	Das Postfach hat die maximale Anzahl für fragmentierte Nachrichten erreicht.
0x50C	1292	0x9811050C	ROUTERERR_FRAGMENTTIMEOUT	Fragment Timeout aufgetreten.
0x50D	1293	0x9811050D	ROUTERERR_TOBEREMOVED	Port wird entfernt.

Allgemeine ADS Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x700	1792	0x98110700	ADSERR_DEVICE_ERROR	Allgemeiner Gerätefehler.
0x701	1793	0x98110701	ADSERR_DEVICE_SRVNOTSUPP	Service wird vom Server nicht unterstützt.
0x702	1794	0x98110702	ADSERR_DEVICE_INVALIDGRP	Ungültige Index-Gruppe.
0x703	1795	0x98110703	ADSERR_DEVICE_INVALIDOFFSET	Ungültiger Index-Offset.
0x704	1796	0x98110704	ADSERR_DEVICE_INVALIDACCESS	Lesen oder Schreiben nicht gestattet.
0x705	1797	0x98110705	ADSERR_DEVICE_INVALIDSIZE	Parametergröße nicht korrekt.
0x706	1798	0x98110706	ADSERR_DEVICE_INVALIDDATA	Ungültige Daten-Werte.
0x707	1799	0x98110707	ADSERR_DEVICE_NOTREADY	Gerät nicht betriebsbereit.
0x708	1800	0x98110708	ADSERR_DEVICE_BUSY	Gerät beschäftigt.
0x709	1801	0x98110709	ADSERR_DEVICE_INVALIDCONTEXT	Ungültiger Kontext vom Betriebssystem - Kann durch Verwendung von ADS Bausteinen in unterschiedlichen Tasks auftreten. Abhilfe kann die Multitasking-Syncronisation in der SPS geben.
0x70A	1802	0x9811070A	ADSERR_DEVICE_NOMEMORY	Nicht genügend Speicher.
0x70B	1803	0x9811070B	ADSERR_DEVICE_INVALIDPARM	Ungültige Parameter-Werte.
0x70C	1804	0x9811070C	ADSERR_DEVICE_NOTFOUND	Nicht gefunden (Dateien,...).
0x70D	1805	0x9811070D	ADSERR_DEVICE_SYNTAX	Syntax-Fehler in Datei oder Befehl.
0x70E	1806	0x9811070E	ADSERR_DEVICE_INCOMPATIBLE	Objekte stimmen nicht überein.
0x70F	1807	0x9811070F	ADSERR_DEVICE_EXISTS	Objekt ist bereits vorhanden.
0x710	1808	0x98110710	ADSERR_DEVICE_SYMBOLNOTFOUND	Symbol nicht gefunden.
0x711	1809	0x98110711	ADSERR_DEVICE_SYMBOLVERSIONINVALID	Symbol-Version ungültig – Kann durch einen Online-Change auftreten. Erzeuge einen neuen Handle.
0x712	1810	0x98110712	ADSERR_DEVICE_INVALIDSTATE	Gerät (Server) ist im ungültigen Zustand.
0x713	1811	0x98110713	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransMode nicht unterstützt.
0x714	1812	0x98110714	ADSERR_DEVICE_NOTIFYHNDINVALID	Notification Handle ist ungültig.
0x715	1813	0x98110715	ADSERR_DEVICE_CLIENTUNKNOWN	Notification-Client nicht registriert.
0x716	1814	0x98110716	ADSERR_DEVICE_NOMOREHDLS	Keine weiteren Handles verfügbar.
0x717	1815	0x98110717	ADSERR_DEVICE_INVALIDWATCHSIZE	Größe der Notification zu groß.
0x718	1816	0x98110718	ADSERR_DEVICE_NOTINIT	Gerät nicht initialisiert.
0x719	1817	0x98110719	ADSERR_DEVICE_TIMEOUT	Gerät hat einen Timeout.
0x71A	1818	0x9811071A	ADSERR_DEVICE_NOINTERFACE	Interface Abfrage fehlgeschlagen.
0x71B	1819	0x9811071B	ADSERR_DEVICE_INVALIDINTERFACE	Falsches Interface angefordert.
0x71C	1820	0x9811071C	ADSERR_DEVICE_INVALIDCLSID	Class-ID ist ungültig.
0x71D	1821	0x9811071D	ADSERR_DEVICE_INVALIDOBJID	Object-ID ist ungültig.
0x71E	1822	0x9811071E	ADSERR_DEVICE_PENDING	Anforderung steht aus.
0x71F	1823	0x9811071F	ADSERR_DEVICE_ABORTED	Anforderung wird abgebrochen.
0x720	1824	0x98110720	ADSERR_DEVICE_WARNING	Signal-Warnung.
0x721	1825	0x98110721	ADSERR_DEVICE_INVALIDARRAYIDX	Ungültiger Array-Index.
0x722	1826	0x98110722	ADSERR_DEVICE_SYMBOLNOTACTIVE	Symbol nicht aktiv.
0x723	1827	0x98110723	ADSERR_DEVICE_ACCESSDENIED	Zugriff verweigert.
0x724	1828	0x98110724	ADSERR_DEVICE_LICENSENOTFOUND	Fehlende Lizenz.
0x725	1829	0x98110725	ADSERR_DEVICE_LICENSEEXPIRED	Lizenz abgelaufen.
0x726	1830	0x98110726	ADSERR_DEVICE_LICENSEEXCEEDED	Lizenz überschritten.
0x727	1831	0x98110727	ADSERR_DEVICE_LICENSEINVALID	Lizenz ungültig.
0x728	1832	0x98110728	ADSERR_DEVICE_LICENSESYSTEMID	Lizenzproblem: System-ID ist ungültig.
0x729	1833	0x98110729	ADSERR_DEVICE_LICENSENOTIMELIMIT	Lizenz nicht zeitlich begrenzt.
0x72A	1834	0x9811072A	ADSERR_DEVICE_LICENSEFUTUREISSUE	Lizenzproblem: Zeitpunkt in der Zukunft.
0x72B	1835	0x9811072B	ADSERR_DEVICE_LICENSETIMETOLONG	Lizenz-Zeitraum zu lang.
0x72C	1836	0x9811072C	ADSERR_DEVICE_EXCEPTION	Exception beim Systemstart.
0x72D	1837	0x9811072D	ADSERR_DEVICE_LICENSEDUPLICATED	Lizenz-Datei zweimal gelesen.
0x72E	1838	0x9811072E	ADSERR_DEVICE_SIGNATUREINVALID	Ungültige Signatur.
0x72F	1839	0x9811072F	ADSERR_DEVICE_CERTIFICATEINVALID	Zertifikat ungültig.
0x730	1840	0x98110730	ADSERR_DEVICE_LICENSEOEMNOTFOUND	Public Key vom OEM nicht bekannt.
0x731	1841	0x98110731	ADSERR_DEVICE_LICENSERESTRICTED	Lizenz nicht gültig für diese System.ID.
0x732	1842	0x98110732	ADSERR_DEVICE_LICENSEDEMODENIED	Demo-Lizenz untersagt.
0x733	1843	0x98110733	ADSERR_DEVICE_INVALIDFNCID	Funktions-ID ungültig.
0x734	1844	0x98110734	ADSERR_DEVICE_OUTOFRANGE	Außerhalb des gültigen Bereiches.
0x735	1845	0x98110735	ADSERR_DEVICE_INVALIDALIGNMENT	Ungültiges Alignment.

Hex	Dec	HRESULT	Name	Beschreibung
0x736	1846	0x98110736	ADSERR_DEVICE_LICENSEPLATFORM	Ungültiger Plattform Level.
0x737	1847	0x98110737	ADSERR_DEVICE_FORWARD_PL	Kontext – Weiterleitung zum Passiv-Level.
0x738	1848	0x98110738	ADSERR_DEVICE_FORWARD_DL	Kontext – Weiterleitung zum Dispatch-Level.
0x739	1849	0x98110739	ADSERR_DEVICE_FORWARD_RT	Kontext – Weiterleitung zur Echtzeit.
0x740	1856	0x98110740	ADSERR_CLIENT_ERROR	Clientfehler.
0x741	1857	0x98110741	ADSERR_CLIENT_INVALIDPARM	Dienst enthält einen ungültigen Parameter.
0x742	1858	0x98110742	ADSERR_CLIENT_LISTEMPTY	Polling-Liste ist leer.
0x743	1859	0x98110743	ADSERR_CLIENT_VARUSED	Var-Verbindung bereits im Einsatz.
0x744	1860	0x98110744	ADSERR_CLIENT_DUPLINVOKEID	Die aufgerufene ID ist bereits in Benutzung.
0x745	1861	0x98110745	ADSERR_CLIENT_SYNCTIMEOUT	Timeout ist aufgetreten – Die Gegenstelle antwortet nicht im vorgegebenen ADS Timeout. Die Routeneinstellung der Gegenstelle kann falsch konfiguriert sein.
0x746	1862	0x98110746	ADSERR_CLIENT_W32ERROR	Fehler im Win32 Subsystem.
0x747	1863	0x98110747	ADSERR_CLIENT_TIMEOUTINVALID	Ungültiger Client Timeout-Wert.
0x748	1864	0x98110748	ADSERR_CLIENT_PORTNOTOPEN	Port nicht geöffnet.
0x749	1865	0x98110749	ADSERR_CLIENT_NOAMSADDR	Keine AMS Adresse.
0x750	1872	0x98110750	ADSERR_CLIENT_SYNCINTERNAL	Interner Fehler in Ads-Sync.
0x751	1873	0x98110751	ADSERR_CLIENT_ADDHASH	Überlauf der Hash-Tabelle.
0x752	1874	0x98110752	ADSERR_CLIENT_REMOVEHASH	Schlüssel in der Tabelle nicht gefunden.
0x753	1875	0x98110753	ADSERR_CLIENT_NOMORESYM	Keine Symbole im Cache.
0x754	1876	0x98110754	ADSERR_CLIENT_SYNCRESINVALID	Ungültige Antwort erhalten.
0x755	1877	0x98110755	ADSERR_CLIENT_SYNCPORTLOCKED	Sync Port ist verriegelt.
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_IRQLNOTLESSOREQUAL	Aufruf einer Service-Funktion im falschen Kontext
0x1017	4119	0x98111017	RTERR_VMXNOTSUPPORTED	Intel VT-x Erweiterung wird nicht unterstützt.
0x1018	4120	0x98111018	RTERR_VMXDISABLED	Intel VT-x Erweiterung ist nicht aktiviert im BIOS.
0x1019	4121	0x98111019	RTERR_VMXCONTROLSMISSING	Fehlende Funktion in Intel VT-x Erweiterung.
0x101A	4122	0x9811101A	RTERR_VMXENABLEFAILS	Aktivieren von Intel VT-x schlägt fehl.

Spezifische positive HRESULT Return Codes:

HRESULT	Name	Beschreibung
0x0000_0000	S_OK	Kein Fehler.
0x0000_0001	S_FALSE	Kein Fehler. Bsp.: erfolgreiche Abarbeitung, bei der jedoch ein negatives oder unvollständiges Ergebnis erzielt wurde.
0x0000_0203	S_PENDING	Kein Fehler. Bsp.: erfolgreiche Abarbeitung, bei der jedoch noch kein Ergebnis vorliegt.
0x0000_0256	S_WATCHDOG_TIMEOUT	Kein Fehler. Bsp.: erfolgreiche Abarbeitung, bei der jedoch eine Zeitüberschreitung eintrat.

TCP Winsock-Fehlercodes

Hex	Dec	Name	Beschreibung
0x274C	10060	WSAETIMEDOUT	Verbindungs Timeout aufgetreten - Fehler beim Herstellen der Verbindung, da die Gegenstelle nach einer bestimmten Zeitspanne nicht ordnungsgemäß reagiert hat, oder die hergestellte Verbindung konnte nicht aufrecht erhalten werden, da der verbundene Host nicht reagiert hat.
0x274D	10061	WSAECONNREFUSED	Verbindung abgelehnt - Es konnte keine Verbindung hergestellt werden, da der Zielcomputer dies explizit abgelehnt hat. Dieser Fehler resultiert normalerweise aus dem Versuch, eine Verbindung mit einem Dienst herzustellen, der auf dem fremden Host inaktiv ist—das heißt, einem Dienst, für den keine Serveranwendung ausgeführt wird.
0x2751	10065	WSAEHOSTUNREACH	Keine Route zum Host - Ein Socketvorgang bezog sich auf einen nicht verfügbaren Host.
Weitere Winsock-Fehlercodes: Win32-Fehlercodes			

8 Support und Service

Beckhoff und seine weltweiten Partnerfirmen bieten einen umfassenden Support und Service, der eine schnelle und kompetente Unterstützung bei allen Fragen zu Beckhoff Produkten und Systemlösungen zur Verfügung stellt.

Downloadfinder

Unser Downloadfinder beinhaltet alle Dateien, die wir Ihnen zum Herunterladen anbieten. Sie finden dort Applikationsberichte, technische Dokumentationen, technische Zeichnungen, Konfigurationsdateien und vieles mehr.

Die Downloads sind in verschiedenen Formaten erhältlich.

Beckhoff Niederlassungen und Vertretungen

Wenden Sie sich bitte an Ihre Beckhoff Niederlassung oder Ihre Vertretung für den lokalen Support und Service zu Beckhoff Produkten!

Die Adressen der weltweiten Beckhoff Niederlassungen und Vertretungen entnehmen Sie bitte unserer Internetseite: www.beckhoff.com

Dort finden Sie auch weitere Dokumentationen zu Beckhoff Komponenten.

Beckhoff Support

Der Support bietet Ihnen einen umfangreichen technischen Support, der Sie nicht nur bei dem Einsatz einzelner Beckhoff Produkte, sondern auch bei weiteren umfassenden Dienstleistungen unterstützt:

- Support
- Planung, Programmierung und Inbetriebnahme komplexer Automatisierungssysteme
- umfangreiches Schulungsprogramm für Beckhoff Systemkomponenten

Hotline:	+49 5246 963-157
E-Mail:	support@beckhoff.com

Beckhoff Service

Das Beckhoff Service-Center unterstützt Sie rund um den After-Sales-Service:

- Vor-Ort-Service
- Reparaturservice
- Ersatzteilservice
- Hotline-Service

Hotline:	+49 5246 963-460
E-Mail:	service@beckhoff.com

Beckhoff Unternehmenszentrale

Beckhoff Automation GmbH & Co. KG

Hülshorstweg 20
33415 Verl
Deutschland

Telefon:	+49 5246 963-0
E-Mail:	info@beckhoff.com
Internet:	www.beckhoff.com

Mehr Informationen:
www.beckhoff.com/te1000

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