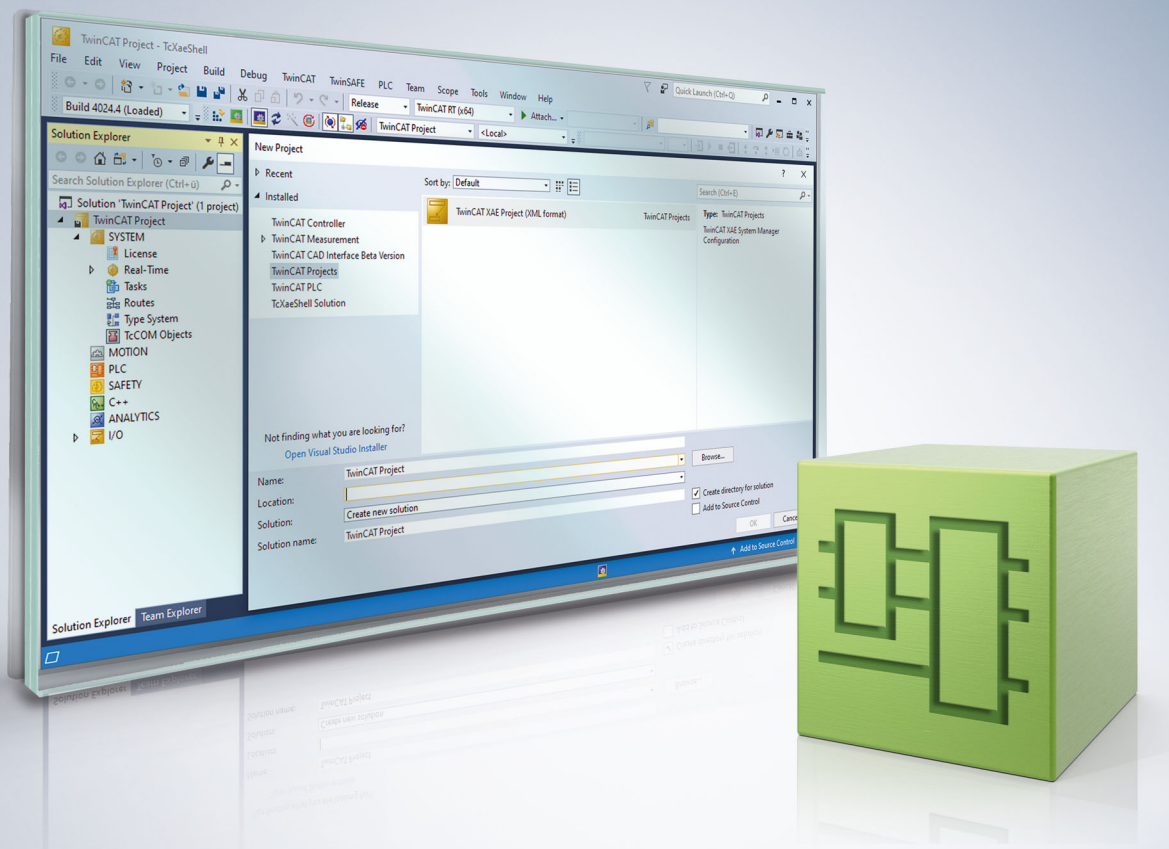
BECKHOFF New Automation Technology

Manual | EN

TE1000

TwinCAT 3 | PLC Lib: Tc3_JsonXml



2024-03-15 | Version: 1.13.0

Table of contents

1 Foreword

1.1 Notes on the documentation

This description is intended exclusively for trained specialists in control and automation technology who are familiar with the applicable national standards.
For installation and commissioning of the components, it is absolutely necessary to observe the documentation and the following notes and explanations.
The qualified personnel is obliged to always use the currently valid documentation.

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

Disclaimer

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

Trademarks

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® and XPlanar® are registered and licensed trademarks of Beckhoff Automation GmbH.
If third parties make use of designations or trademarks used in this publication for their own purposes, this could infringe upon the rights of the owners of the said designations.

Patents

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

EtherCAT®

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

Copyright

1.2 For your safety

Safety regulations

Read the following explanations for your safety.
Always observe and follow product-specific safety instructions, which you may find at the appropriate places in this document.

Exclusion of liability

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

Personnel qualification

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

Signal words

The signal words used in the documentation are classified below. In order to prevent injury and damage to persons and property, read and follow the safety and warning notices.

Personal injury warnings

⚠ DANGER

Hazard with high risk of death or serious injury.

⚠ WARNING

Hazard with medium risk of death or serious injury.

⚠ CAUTION

There is a low-risk hazard that could result in medium or minor injury.

Warning of damage to property or environment

NOTICE

The environment, equipment, or data may be damaged.

Information on handling the product

This information includes, for example:
recommendations for action, assistance or further information on the product.

1.3 Notes on information security

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

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

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

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

1.4 Documentation issue status

Version	Modification
1.13.0	New: Function blocks/FB_JsonDomParser/PushbackCopyValue

2 Overview

With the aid of the PLC library Tc3_JsonXml, SAX and DOM parser technologies can be used to create and navigate through JSON and XML documents.

System requirements

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 was originally developed for handling XML documents, but can also be used for other data formats such as JSON. A SAX parser treats the data to be read or written as a sequential data stream. When reading a data stream, defined callback methods are called, which then return the corresponding contents of the data stream. A SAX parser is therefore also referred to as an event-based parser. The events occurring (callback methods) are stateless, i.e. they are not dependent on the preceding events. The advantage of this is that the XML document never has to be completely contained in the memory and the application can react "on the fly" via the callbacks.

DOM (Document Object Model)

DOM is a specification for accessing XML documents, but can also be used for other data formats such as HTML or JSON. The interface is based on a defined object model, whose validity is a prerequisite for correct use. This object model represents a document, e.g. a JSON document, in the form of a tree structure in the memory, which can then be used to navigate through the document. DOM allows navigation between the individual nodes, the creation, moving and deletion of nodes, and the reading, changing and deletion of node contents. When editing is finished a new JSON or XML document is generated from the finalized tree structure. The advantage here is that no own data housekeeping needs to be created with the read-in data, since the data exist in the DOM and can be continuously accessed.

JSON document

The following section shows a JSON document as an example:

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

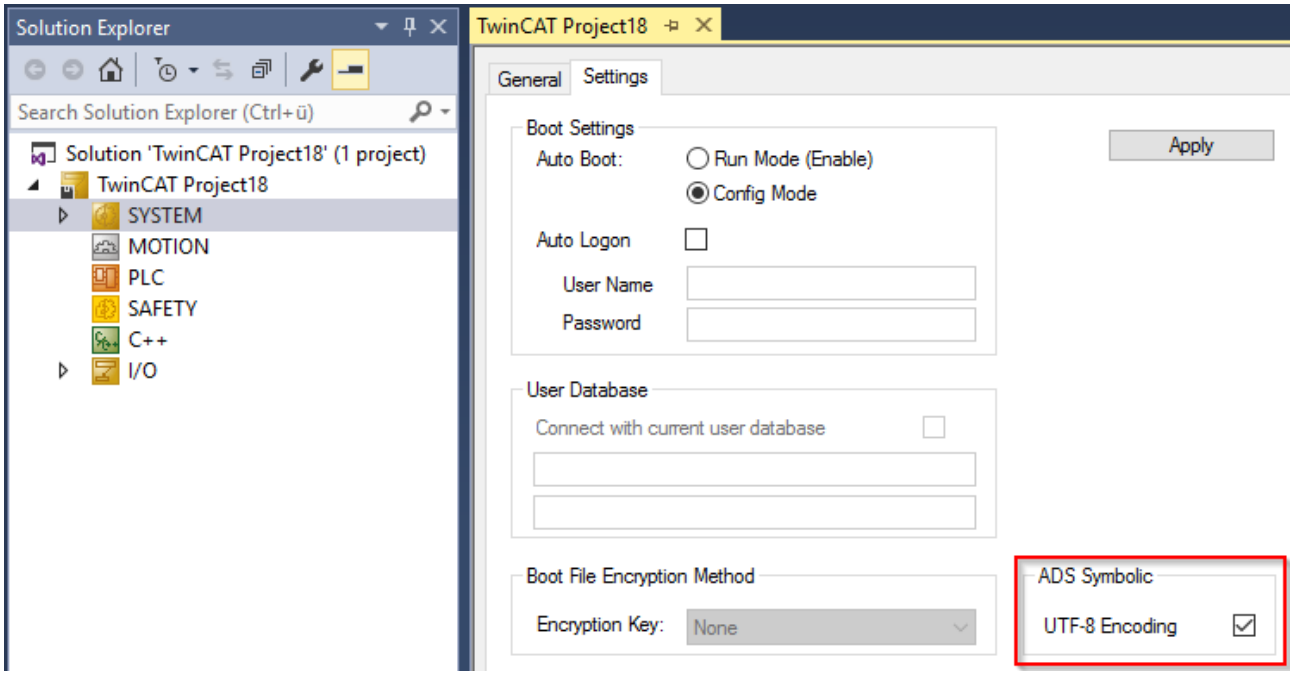
Metadata

The Tc3_JsonXml library contains the function block FB_JsonReadWriteDataType [▶ 132], which enables automatic generation of metadata by means of PLC attributes.

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

See also: Examples > Tc3JsonXmlSampleJsonDataType [▶ 216].

In order to use UTF-8 characters, e.g. in the automatic generation of metadata via the function block FB_JsonReadWriteDataType [▶ 132], the check box for the support of UTF-8 in the symbolism must be activated in the TwinCAT project. To do this, double-click on **SYSTEM** in the project tree, open the **Settings** tab and activate the corresponding check box.



JSON Web Token (JWT)

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

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

The Tc3_JsonXml library provides a function for creating a JWT via the FB_JwtEncode [▶ 209] method.

3 Getting started

This documentation article is intended as a quick-start guide to using the Tc3_JsonXml PLC library. The article is based on the following use case:

- A JSON document exists that is to be processed further.
- The JSON document is in the form of a file on the local file system.
- The article describes the steps to parse the JSON document.

For more code samples, please refer to our samples [▶ 216].

File contents

The file *myJsonContent.json* is located in the local file system under the following path: *c:\temp\myJsonContent.json* and has the following content:

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

Declarations

The following variable declarations are required for the further procedure:

```
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;
```

Furthermore, please make sure that you have added a reference to the PLC library Tc3_JsonXml in your PLC project.

Reading the file

The method LoadDocumentFromFile [▶ 62] from the function block FB_JsonDomParser [▶ 19] is used to read the file. The execution of the method is controlled by a rising edge at input `bExec`.

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

Source of the JSON document

If in your use case the JSON document already exists in the PLC, e.g. in a variable of data type STRING, you can use the method ParseDocument [▶ 65] to load the document into memory and for further processing.

Parsing the JSON document

The GetDocumentRoot [▶ 46] method can be used to reference the beginning of the JSON document in memory. The return value of the method is an interface pointer.

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

From here, the other keys on the first level can be read, e.g. the 'Timestamp' key via the method GetString [▶ 51]:

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

The next key ('Values') is a nested JSON object. The child elements of this object can be read directly via the method FindMemberPath [▶ 40], for example for the element 'Values/Sensor1':

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

The next child element ('Values/Sensor2') is an array. This can be read by using the methods ArrayBegin [▶ 34] and ArrayEnd [▶ 35].

```
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
```

Handling of the next child element ('Values/Sensor3') is similar to the element 'Values/Sensor1'. Instead of the method GetDouble [▶ 46] the method GetInt [▶ 48] is used to read the value of the key.

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

Complete code sample

Below you will find the above code snippets as a complete sample.

Declaration part

```
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
```

Implementation part

```
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
```

After the above code has been executed successfully, the keys that were read out are in the corresponding PLC variables, e.g. as follows:

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 Function blocks

4.1 FB_JsonDomParser

This function block is derived from the same internal function block as FB_JsonDynDomParser [▶ 88] and thus offers the same interface.

The two derived function blocks differ only in their internal memory management. FB_JsonDomParser is optimized for the fast and efficient parsing and creation of JSON documents that are only changed a little. The function block FB_JsonDynDomParser [▶ 88] is recommended for JSON documents to which many changes are made (e.g. the cyclic changing of a specific value in the JSON document).

⚠ WARNING

Use of router memory

The function block occupies new memory with each change, e.g. with the methods SetObject() or SetJson(). As a result, the amount of router memory used can grow enormously after repeated actions. This allocated memory is only released again by calling the NewDocument [▶ 64]() or ParseDocument [▶ 65]() methods.

Strings in UTF-8 format

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

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

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

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

Syntax

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

Outputs

Name	Type
initStatus	HRESULT

Methods

Name	Description
AddArrayMember [▶ 23]	Adds an Array member to a JSON object.
AddBase64Member [▶ 24]	Adds a Base64 member to a JSON object.
AddBoolMember [▶ 25]	Adds a Bool member to a JSON object.
AddDateTimeMember [▶ 26]	Adds a DateTime member to a JSON object.
AddDcTimeMember [▶ 26]	Adds a DcTime member to a JSON object.
AddDoubleMember [▶ 27]	Adds a Double member to a JSON object.
AddFileTimeMember [▶ 28]	Adds a FileTime member to a JSON object.
AddHexBinaryMember [▶ 28]	Adds a HexBinary member to a JSON object.
AddInt64Member [▶ 29]	Adds an Int64 member to a JSON object.
AddIntMember [▶ 30]	Adds an Int member to a JSON object.
AddJsonMember [▶ 30]	Adds a JSON member to a JSON object.
AddNullMember [▶ 31]	Adds a NULL member to a JSON object.
AddObjectMember [▶ 32]	Adds an Object member to a JSON object.
AddStringMember [▶ 32]	Adds a String member to a JSON object.
AddUint64Member [▶ 33]	Adds an UInt64 member to a JSON object.
AddUintMember [▶ 34]	Adds an UInt member to a JSON object.
ArrayBegin [▶ 34]	Returns the first element of an array.
ArrayEnd [▶ 35]	Returns the last element of an array.
ClearArray [▶ 36]	Deletes the contents of an array.
CopyDocument [▶ 36]	Copies the contents of the DOM memory into a variable of data type STRING.
CopyJson [▶ 37]	Extracts a JSON object from a key and stores it in a variable of data type STRING.
CopyString [▶ 38]	Copies the value of a key into a variable of data type STRING.
FindMember [▶ 40]	Searches for a specific property in a JSON document.
FindMemberPath [▶ 40]	Searches for a specific property in a JSON document (path-specific).
GetArraySize [▶ 42]	Returns the number of elements in a JSON array.
GetArrayValue [▶ 42]	Returns the value at the current iterator position of an array.
GetArrayValueByIdx [▶ 43]	Returns the value of an array at a specified index.
GetBase64 [▶ 43]	Decodes a Base64 value from a JSON property.
GetBool [▶ 44]	Returns the value of a property of the data type BOOL.
GetDateTime [▶ 44]	Returns the value of a property of the data type DATE_AND_TIME.
GetDcTime [▶ 45]	Returns the value of a property of the data type DCTIME.
GetDocument [▶ 45]	Returns the content of the DOM memory as the data type STRING(255).
GetDocumentLength [▶ 46]	Returns the length of a JSON document in the DOM memory.
GetDocumentRoot [▶ 46]	Returns the root node of a JSON document in the DOM memory.
GetDouble [▶ 46]	Returns the value of a property of the data type LREAL.
GetFileTime [▶ 47]	Returns the value of a property of the data type DCTIME.
GetHexBinary [▶ 47]	Decodes the HexBinary content of a property and writes it to a certain memory address,
GetInt [▶ 48]	Returns the value of a property of the data type DINT.
GetInt64 [▶ 48]	Returns the value of a property of the data type LINT.
GetJson [▶ 49]	Returns the value of a property of the data type STRING(255).
GetJsonLength [▶ 49]	Returns the length of a property if this is a JSON document.

Name	Description
GetMaxDecimalPlaces [▶ 50]	Returns the current setting for MaxDecimalPlaces.
GetMemberName [▶ 50]	Returns the name of a JSON property member at the position of the current iterator,
GetMemberValue [▶ 51]	Returns the value of a JSON property member at the position of the current iterator,
GetString [▶ 51]	Returns the value of a property of the data type STRING(255).
GetStringLength [▶ 52]	Returns the length of a property if its value is a string.
GetType [▶ 52]	Returns the type of a property value.
GetUint [▶ 53]	Returns the value of a property of the data type UDINT.
GetUint64 [▶ 54]	Returns the value of a property of the data type ULINT.
HasMember [▶ 54]	Checks whether a certain property is present in the DOM memory.
IsArray [▶ 55]	Checks whether a given property is an array.
IsBase64 [▶ 55]	Checks whether the value of a given property is of the data type Base64.
IsBool [▶ 56]	Checks whether the value of a given property is of the data type BOOL.
IsDouble [▶ 56]	Checks whether the value of a given property is of the data type Double (PLC: LREAL).
IsFalse [▶ 56]	Checks whether the value of a given property is FALSE.
IsHexBinary [▶ 57]	Checks whether the value of a property is in the HexBinary format.
IsInt [▶ 57]	Checks whether the value of a given property is of the data type Integer (PLC: DINT) .
IsInt64 [▶ 58]	Checks whether the value of a given property is of the data type LINT.
IsISO8601TimeFormat [▶ 58]	Checks whether the value of a given property has a time format according to ISO8601.
IsNull [▶ 59]	Checks whether the value of a given property is NULL.
IsNumber [▶ 59]	Checks whether the value of a given property is a numerical value.
IsObject [▶ 60]	Checks whether the given property is a further JSON object.
IsString [▶ 60]	Checks whether the value of a given property is of the data type STRING.
IsTrue [▶ 61]	Checks whether the value of a given property is TRUE.
IsUint [▶ 61]	Checks whether the value of a given property is of the data type UDINT.
IsUint64 [▶ 61]	Checks whether the value of a given property is of the data type ULINT.
LoadDocumentFromFile [▶ 62]	Loads a JSON document from a file.
MemberBegin [▶ 63]	Returns the first child element below a JSON property.
MemberEnd [▶ 63]	Returns the last child element below a JSON property.
NewDocument [▶ 64]	Generates a new empty JSON document in the DOM memory.
NextArray [▶ 64]	Returns the next element in an array.
NextMember	Returns the next property in a JSON document.
ParseDocument [▶ 65]	Loads a JSON object into the DOM memory for further processing.
PopbackValue	Deletes the element at the end of an array.
PushbackBase64Value [▶ 65]	Appends a Base64 value to the end of an array.
PushbackBoolValue [▶ 66]	Appends a Base64 value to the end of an array.
PushbackDateTimeValue [▶ 67]	Appends a value of the data type DATE_AND_TIME to the end of an array.
PushbackDcTimeValue [▶ 67]	Appends a value of the data type DCTIME to the end of an array.

Name	Description
PushbackDoubleValue [▶ 68]	Appends a value of the data type Double to the end of an array.
PushbackFileTimeValue [▶ 69]	Appends a value of the data type FILETIME to the end of an array.
PushbackHexBinaryValue [▶ 69]	This method appends a HexBinary-coded value to the end of an array.
PushbackInt64Value [▶ 70]	Appends a value of the data type Int64 to the end of an array.
PushbackIntValue [▶ 70]	Appends a value of the data type INT to the end of an array.
PushbackJsonValue [▶ 71]	Appends a JSON document to the end of an array.
PushbackNullValue [▶ 72]	Appends a NULL value to the end of an array.
PushbackStringValue [▶ 72]	Appends a value of the data type String to the end of an array.
PushbackUint64Value [▶ 73]	Appends a value of the data type UInt64 to the end of an array.
PushbackUintValue [▶ 73]	Appends a value of the data type UInt to the end of an array.
RemoveAllMembers [▶ 74]	Removes all child elements from a given property.
RemoveArray [▶ 74]	Deletes the value of the current array iterator.
RemoveMember [▶ 75]	Deletes the property at the current iterator.
RemoveMemberByName [▶ 76]	Removes a child element from a given property.
SaveDocumentToFile [▶ 77]	Saves a JSON document in a file.
SetArray [▶ 78]	Sets the value of a property to the type "Array".
SetBase64 [▶ 78]	Sets the value of a property to a Base64-coded value.
SetBool [▶ 79]	Sets the value of a property to a value of the data type BOOL.
SetDateTime [▶ 80]	Sets the value of a property to a value of the data type DATE_AND_TIME.
SetDcTime [▶ 80]	Sets the value of a property to a value of the data type DCTIME.
SetDouble [▶ 81]	Sets the value of a property to a value of the data type Double.
SetFileTime [▶ 81]	Sets the value of a property to a value of the data type FILETIME.
SetHexBinary [▶ 82]	Sets the value of a property to a HexBinary-coded value.
SetInt [▶ 82]	Sets the value of a property to a value of the data type INT
SetInt64 [▶ 83]	Sets the value of a property to a value of the data type Int64.
SetJson [▶ 83]	Inserts a further JSON document into the value of a property.
SetMaxDecimalPlaces [▶ 84]	Sets the current setting for MaxDecimalPlaces.
SetNull [▶ 84]	Sets the value of a property to the value NULL.
SetObject [▶ 85]	Sets the value of a property to the type "Object".
SetString [▶ 85]	Sets the value of a property to a value of the data type STRING.
SetUint [▶ 86]	Sets the value of a property to a value of the data type UInt.
SetUint64 [▶ 87]	Sets the value of a property to a value of the data type UInt64.

Requirements

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

4.1.1 AddArrayMember

This method adds an array member to a JSON object.

Syntax

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

Return value

Name	Type
AddArrayMember	SJsonValue

Inputs

Name	Type
v	SJsonValue
reserve	UDINT

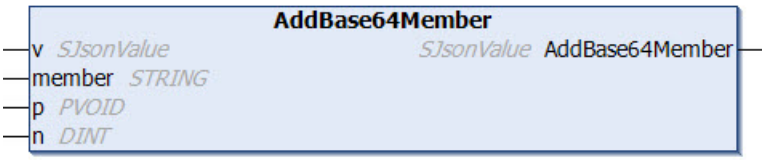
/ Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

4.1.2 AddBase64Member



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

Syntax

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

Return value

Name	Type
AddBase64Member	SJsonValue

Inputs

Name	Type
v	SJsonValue
p	PVOID
n	DINT

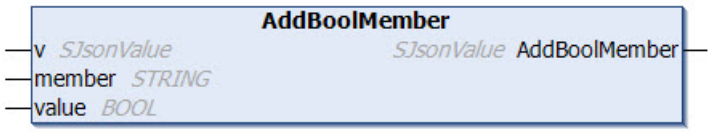
/ Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

4.1.3 AddBoolMember



This method adds a Bool member to a JSON object.

Syntax

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

Return value

Name	Type
AddBoolMember	SJsonValue

Inputs

Name	Type
v	SJsonValue
value	BOOL

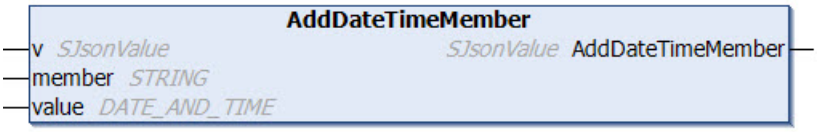
/ Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

4.1.4 AddDateTimeMember



This method adds a DateTime member to a JSON object.

Syntax

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

Return value

Name	Type
AddDateTimeMember	SJsonValue

Inputs

Name	Type
v	SJsonValue
value	DATE_AND_TIME

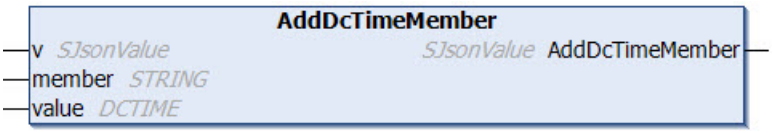
/ Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

4.1.5 AddDcTimeMember



This method adds a DcTime member to a JSON object.

Syntax

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

Return value

Name	Type
AddDcTimeMember	SJsonValue

Inputs

Name	Type
v	SJsonValue
value	DCTIME

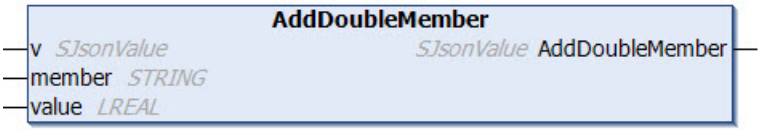
Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

4.1.6 AddDoubleMember



This method adds a Double member to a JSON object.

Syntax

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

Return value

Name	Type
AddDoubleMember	SJsonValue

Inputs

Name	Type
v	SJsonValue
value	LREAL

Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

4.1.7 AddFileTimeMember

This method adds a FileTime member to a JSON object.

Syntax

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

Return value

Name	Type
AddFileTimeMember	SJsonValue

Inputs

Name	Type
v	SJsonValue
value	FILETIME

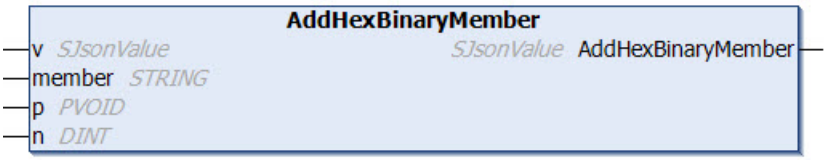
/ Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

4.1.8 AddHexBinaryMember



This method adds a HexBinary member to a JSON object.

Syntax

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

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

Return value

Name	Type
AddHexBinaryMember	SjsonValue

Inputs

Name	Type
v	SjsonValue
p	PVOID
n	DINT

/ Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

4.1.9 AddInt64Member

This method adds an Int64 member to a JSON object.

Syntax

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

Return value

Name	Type
AddInt64Member	SJsonValue

Inputs

Name	Type
v	SJsonValue
value	LINT

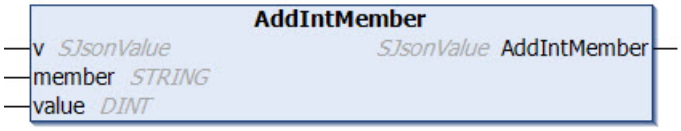
Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

4.1.10 AddIntMember



This method adds an Int member to a JSON object.

Syntax

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

Return value

Name	Type
AddIntMember	SJsonValue

Inputs

Name	Type
v	SJsonValue
value	DINT

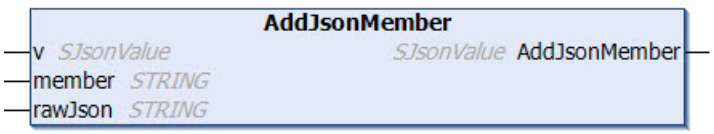
Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

4.1.11 AddJsonMember



This method adds a JSON member to a JSON object.

Syntax

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

Return value

Name	Type
AddJsonMember	SJsonValue

Inputs

Name	Type
v	SJsonValue

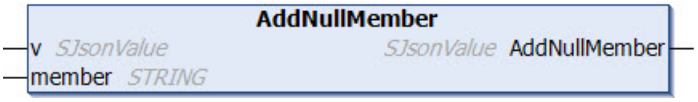
/ Inputs/Outputs

Name	Type
member	STRING
rawJson	STRING

Sample call:

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

4.1.12 AddNullMember



This method adds a NULL member to a JSON object.

Syntax

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

Return value

Name	Type
AddNullMember	SJsonValue

Inputs

Name	Type
v	SJsonValue

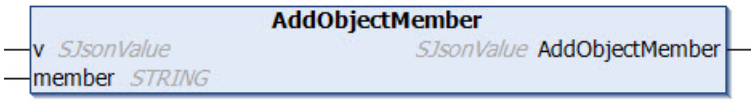
Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

4.1.13 AddObjectMember



This method adds an Object member to a JSON object.

Syntax

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

Return value

Name	Type
AddObjectMember	SJsonValue

Inputs

Name	Type
v	SJsonValue

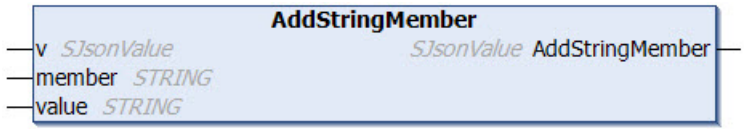
Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

4.1.14 AddStringMember



This method adds a String member to a JSON object.

Syntax

```
METHOD AddStringMember : SJsonValue
VAR_INPUT
  v       : SJsonValue;
```

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

Return value

Name	Type
AddStringMember	SJsonValue

Inputs

Name	Type
v	SJsonValue

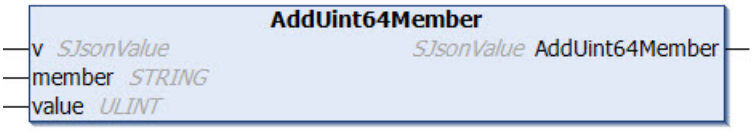
/ Inputs/Outputs

Name	Type
member	STRING
value	STRING

Sample call:

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

4.1.15 AddUint64Member



This method adds an UInt64 member to a JSON object.

Syntax

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

Return value

Name	Type
AddUint64Member	SJsonValue

Inputs

Name	Type
v	SJsonValue
value	ULINT

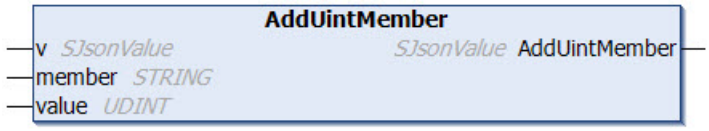
Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

4.1.16 AddUintMember



This method adds an UInt member to a JSON object.

Syntax

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

Return value

Name	Type
AddUintMember	SJsonValue

Inputs

Name	Type
v	SJsonValue
value	UDINT

Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

4.1.17 ArrayBegin

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

Syntax

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

Return value

Name	Type
ArrayBegin	SJsonAIterator

Inputs

Name	Type
v	SJsonValue

Sample call:

```
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

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

Syntax

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

Return value

Name	Type
ArrayEnd	SJsonAIterator

Inputs

Name	Type
v	SJsonValue

Sample call:

```
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

This method deletes the content of an array.

Syntax

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

Return value

Name	Type
ClearArray	BOOL

Inputs

Name	Type
v	SJsonValue
i	SJsonAIterator

Sample call:

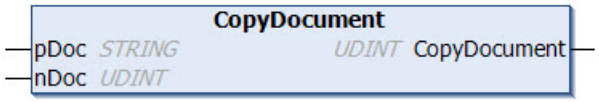
The following JSON document is to be loaded into the DOM memory:

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

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

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

4.1.20 CopyDocument



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

Syntax

```
METHOD CopyDocument : UDINT
VAR_INPUT
  nDoc : DINT;
```

```
END_VAR
VAR_IN_OUT CONSTANT
  pDoc : STRING;
END_VAR
```

Return value

Name	Type
CopyDocument	UDINT

Inputs

Name	Type
nDoc	DINT

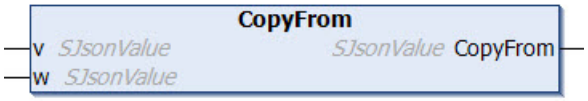
/ Inputs/Outputs

Name	Type
pDoc	STRING

Sample call:

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

4.1.21 CopyFrom



Syntax

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

Return value

Name	Type
CopyFrom	SJsonValue

Inputs

Name	Type
v	SJsonValue
w	SJsonValue

4.1.22 CopyJson

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

Syntax

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

Return value

Name	Type
CopyJson	UDINT

Inputs

Name	Type
v	SJsonValue

Inputs/Outputs

Name	Type
pDoc	STRING
nDoc	STRING

Sample call:

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

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

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

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

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

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

4.1.23 CopyString

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

Syntax

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

Return value

Name	Type
CopyString	UDINT

Inputs

Name	Type
v	SJsonValue

Inputs/Outputs

Name	Type
pStr	STRING
nStr	UDINT

Sample call:

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

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

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

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

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

```
SINGLE
```

4.1.24 ExceptionRaised

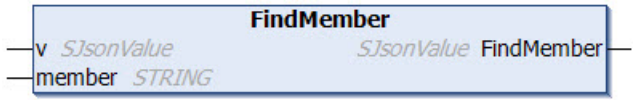
Syntax

```
METHOD ExceptionRaised : BOOL
```

Return value

Name	Type
ExceptionRaised	BOOL

4.1.25 FindMember



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

Syntax

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

Return value

Name	Type
FindMember	SJsonValue

Inputs

Name	Type
v	SJsonValue

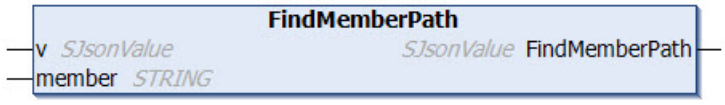
/ Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

4.1.26 FindMemberPath



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

Syntax

```
METHOD FindMemberPath : SJsonValue
VAR_INPUT
  v      : SJsonValue
END_VAR
```

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

Return value

Name	Type
FindMemberPath	SJsonValue

Inputs

Name	Type
v	SJsonValue

/ Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

Accessing nested objects works according to the scheme *a/b/c*, to find a variable in a JSON hierarchy. The call for the variable *c* of the following JSON document is:

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

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

Support for arrays

The method supports JSON documents with arrays from TwinCAT version >3.1.4024.35. The # character can be used to access elements of an array.

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

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

The example call accesses the second element of the outer array (#1), then the third element of the array under the sub-element *Third*.

Treatment of special cases

Inserting the ~ character provides special treatment within a path. The following table lists the different possibilities.

Expression	Result	Sample
~0	~ is used as a character in the string.	Test/Hello~0123 becomes Test/Hello~123
~1	The expression is replaced by the character /, which is not interpreted as a separator, but as part of the string.	Test/Hello~1123 becomes Test/Hello/123
~2	The expression is replaced by the character # , which is not interpreted as an array index, but as part of the string.	Test/Hello~2123 becomes Test/Hello#123

4.1.27 GetArraySize

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

Syntax

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

Return value

Name	Type
GetArraySize	UDINT

Inputs

Name	Type
v	SJsonValue

Sample call:

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

4.1.28 GetArrayValue

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

Syntax

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

Return value

Name	Type
GetArrayValue	SJsonValue

Inputs

Name	Type
i	SJsonAIterator

Sample call:

```
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

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

Syntax

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

Return value

Name	Type
GetArrayValueByIdx	SJsonValue

Inputs

Name	Type
v	SJsonValue
idx	UDINT

Sample call:

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

4.1.30 GetBase64

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

Syntax

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

Return value

Name	Type
GetBase64	DINT

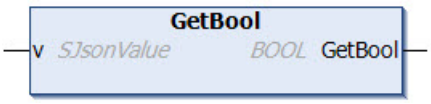
Inputs

Name	Type
v	SJsonValue
p	PVOID
n	DINT

Sample call:

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

4.1.31 GetBool



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

Syntax

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

Return value

Name	Type
GetBool	BOOL

Inputs

Name	Type
v	SJsonValue

4.1.32 GetDateTime

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

Syntax

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

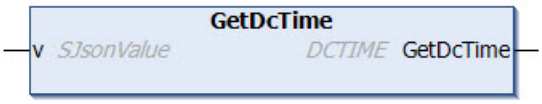
Return value

Name	Type
GetDateTime	DATE_AND_TIME

Inputs

Name	Type
v	SJsonValue

4.1.33 GetDcTime



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

Syntax

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

Return value

Name	Type
GetDcTime	DCTIME

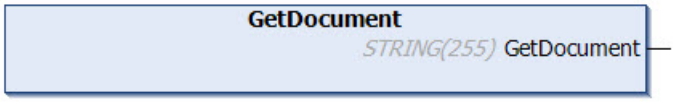
Inputs

Name	Type
v	SJsonValue

Sample call:

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

4.1.34 GetDocument



This method returns the content of the DOM memory as the data type STRING(255). With longer strings, the method will return a NULL string. In this case the method CopyDocument [▶ 36]() must be used.

Syntax

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

Return value

Name	Type
GetDocument	STRING(255)

Sample call:

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

4.1.35 GetDocumentLength

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

Syntax

```
METHOD GetDocumentLength: UDINT
```

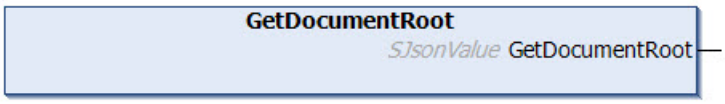
Return value

Name	Type
GetDocumentLength	UDINT

Sample call:

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

4.1.36 GetDocumentRoot



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

Syntax

```
METHOD GetDocumentRoot : SJsonValue
```

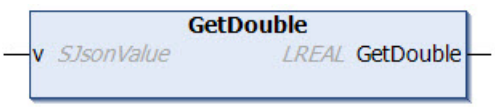
Return value

Name	Type
GetDocumentRoot	SJsonValue

Sample call:

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

4.1.37 GetDouble



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

Syntax

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

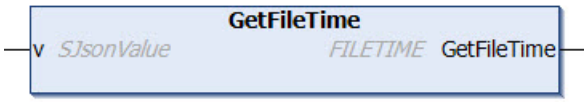
Return value

Name	Type
GetDouble	LREAL

Inputs

Name	Type
v	SJsonValue

4.1.38 GetFileTime



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

Syntax

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

Return value

Name	Type
GetFileTime	FILETIME

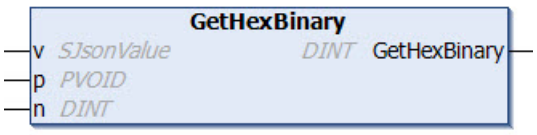
Inputs

Name	Type
v	SJsonValue

Sample call:

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

4.1.39 GetHexBinary



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

Syntax

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

Return value

Name	Type
GetHexBinary	DINT

Inputs

Name	Type
v	SJsonValue
p	PVOID
n	DINT

Sample call:

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

4.1.40 GetInt

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

Syntax

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

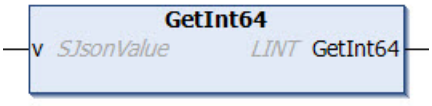
Return value

Name	Type
GetInt	DINT

Inputs

Name	Type
v	SJsonValue

4.1.41 GetInt64



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

Syntax

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

Return value

Name	Type
GetInt64	LINT

Inputs

Name	Type
v	SJsonValue

4.1.42 GetJson

This method returns the value of a property as data type STRING(255), if this is a JSON document itself. With longer strings, the method will return a NULL string. In this case the method CopyJson [▶ 37]() must be used.

Syntax

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

Return value

Name	Type
GetJson	STRING(255)

Inputs

Name	Type
v	SJsonValue

Sample call:

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

4.1.43 GetJsonLength

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

Syntax

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

Return value

Name	Type
GetJsonLength	UDINT

Inputs

Name	Type
v	SJsonValue

Sample call:

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

4.1.44 GetMaxDecimalPlaces

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

Syntax

```
METHOD GetMaxDecimalPlaces : DINT
```

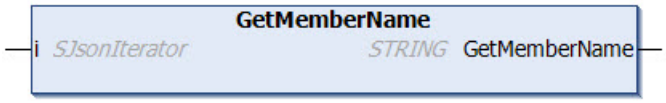
Return value

Name	Type
GetMaxDecimalPlaces	DINT

Sample call:

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

4.1.45 GetMemberName



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

Syntax

```
METHOD GetMemberName : STRING(255)
VAR_INPUT
  i : SJsonIterator;
END_VAR
```

Return value

Name	Type
GetMemberName	STRING(255)

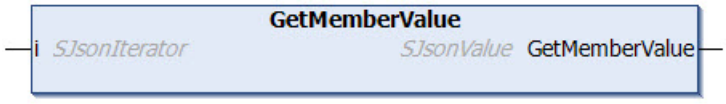
Inputs

Name	Type
i	SJsonIterator

Sample call:

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

4.1.46 GetMemberValue



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

Syntax

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

Return value

Name	Type
GetMemberValue	SJsonValue

Inputs

Name	Type
i	SJsonIterator

Sample call:

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

4.1.47 GetString

This method returns the value of a property of the data type STRING(255). With longer strings, the method will return a NULL string. In this case the method CopyString [▶ 38]() must be used.

Syntax

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

Return value

Name	Type
GetString	STRING(255)

Inputs

Name	Type
v	SJsonValue

4.1.48 GetStringLength

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

Syntax

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

Return value

Name	Type
GetStringLength	UDINT

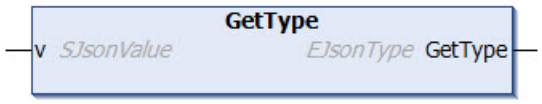
Inputs

Name	Type
v	SJsonValue

Sample call:

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

4.1.49 GetType



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

Syntax

```
METHOD 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;
```

Return value

Name	Type
GetType	EJsonType

Inputs

Name	Type
v	SJsonValue

Sample call:

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

4.1.50 GetUint

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

Syntax

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

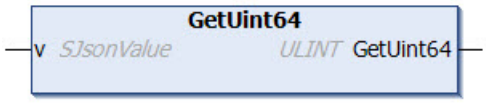
Return value

Name	Type
GetUint	UDINT

Inputs

Name	Type
v	SJsonValue

4.1.51 GetUint64



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

Syntax

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

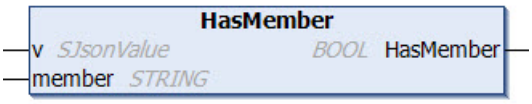
Return value

Name	Type
GetUint64	ULINT

Inputs

Name	Type
v	SJsonValue

4.1.52 HasMember



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

Syntax

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

Return value

Name	Type
HasMember	BOOL

Inputs

Name	Type
v	SJsonValue

/ Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

4.1.53 IsArray

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

Syntax

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

Return value

Name	Type
IsArray	BOOL

Inputs

Name	Type
v	SJsonValue

4.1.54 IsBase64

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

Syntax

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

Return value

Name	Type
IsBase64	BOOL

Inputs

Name	Type
v	SJsonValue

4.1.55 IsBool

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

Syntax

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

Return value

Name	Type
IsBool	BOOL

Inputs

Name	Type
v	SJsonValue

4.1.56 IsDouble

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

Syntax

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

Return value

Name	Type
IsDouble	BOOL

Inputs

Name	Type
v	SJsonValue

4.1.57 IsFalse

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

Syntax

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

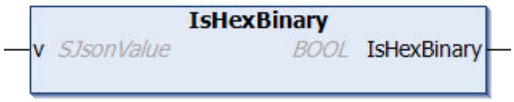
Return value

Name	Type
IsFalse	BOOL

Inputs

Name	Type
v	SJsonValue

4.1.58 IsHexBinary



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

Syntax

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

Return value

Name	Type
IsHexBinary	BOOL

Inputs

Name	Type
v	SJsonValue

Sample call:

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

4.1.59 IsInt

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

Syntax

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

Return value

Name	Type
IsInt	BOOL

Inputs

Name	Type
v	SJsonValue

4.1.60 IsInt64

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

Syntax

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

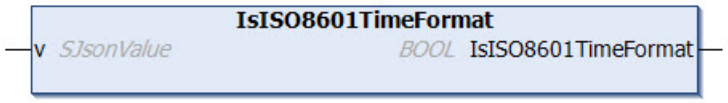
Return value

Name	Type
IsInt64	BOOL

Inputs

Name	Type
v	SJsonValue

4.1.61 IsISO8601TimeFormat



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

Syntax

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

Return value

Name	Type
IsISO8601TimeFormat	BOOL

Inputs

Name	Type
v	SJsonValue

4.1.62 IsNull

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

Syntax

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

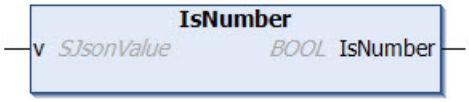
Return value

Name	Type
IsNull	BOOL

Inputs

Name	Type
v	SJsonValue

4.1.63 IsNumber



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

Syntax

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

Return value

Name	Type
IsNumber	BOOL

Inputs

Name	Type
v	SJsonValue

4.1.64 IsObject

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

Syntax

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

Return value

Name	Type
IsObject	BOOL

Inputs

Name	Type
v	SJsonValue

4.1.65 IsString

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

Syntax

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

Return value

Name	Type
IsString	BOOL

Inputs

Name	Type
v	SJsonValue

4.1.66 IsTrue

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

Syntax

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

Return value

Name	Type
IsTrue	BOOL

Inputs

Name	Type
v	SJsonValue

4.1.67 IsUint

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

Syntax

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

Return value

Name	Type
IsUint	BOOL

Inputs

Name	Type
v	SJsonValue

4.1.68 IsUint64

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

Syntax

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

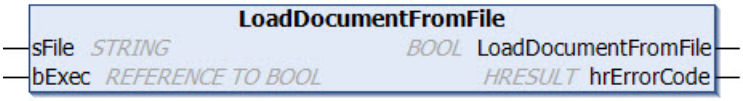
Return value

Name	Type
IsUint64	BOOL

Inputs

Name	Type
v	SJsonValue

4.1.69 LoadDocumentFromFile



This method loads a JSON document from a file.

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

Syntax

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

Return value

Name	Type
LoadDocumentFromFile	BOOL

Inputs

Name	Type
bExec	REFERENCE TO BOOL

Inputs/Outputs

Name	Type
sFile	STRING

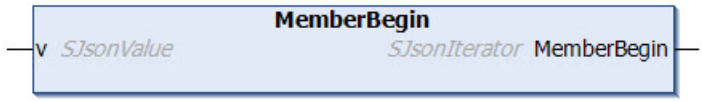
Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

4.1.70 MemberBegin



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

Syntax

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

Return value

Name	Type
MemberBegin	SJsonIterator

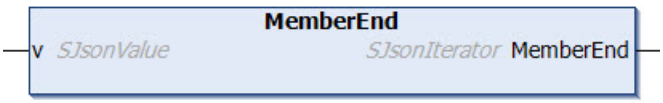
Inputs

Name	Type
v	SJsonValue

Sample call:

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

4.1.71 MemberEnd



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

Syntax

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

Return value

Name	Type
MemberEnd	SJsonIterator

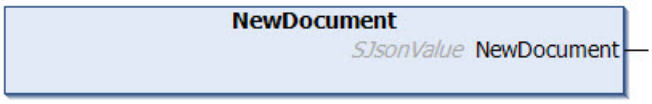
Inputs

Name	Type
v	SJsonValue

Sample call:

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

4.1.72 NewDocument



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

Syntax

```
METHOD NewDocument : SJsonValue
```

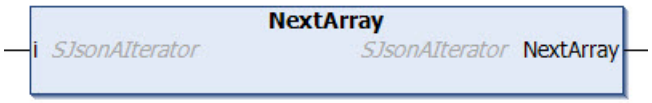
Sample call:

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

Return value

Name	Type
NewDocument	SJsonValue

4.1.73 NextArray



Syntax

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

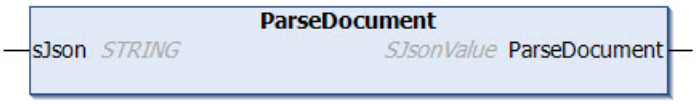
Return value

Name	Type
NextArray	SJsonAIterator

Inputs

Name	Type
i	SJsonAlterator

4.1.74 ParseDocument



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

Syntax

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

Return value

Name	Type
ParseDocument	SJsonValue

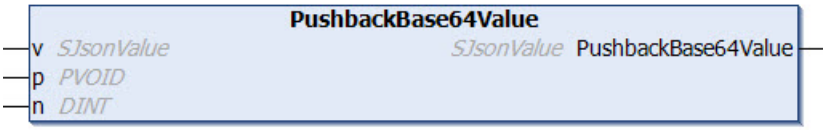
Inputs

Name	Type
sJson	STRING

Sample call:

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

4.1.75 PushbackBase64Value



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

Syntax

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

Return value

Name	Type
PushbackBase64Value	SJsonValue

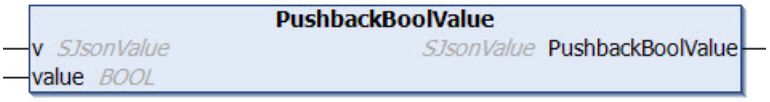
Inputs

Name	Type
v	sJsonValue
p	PVOID
n	DINT

Sample call:

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

4.1.76 PushbackBoolValue



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

Syntax

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

Return value

Name	Type
PushbackBoolValue	SJsonValue

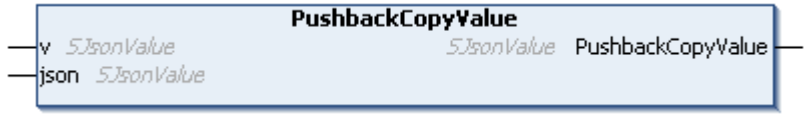
Inputs

Name	Type
v	sJsonValue
value	BOOL

Sample call:

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

4.1.77 PushbackCopyValue



This method adds a JSON document from the memory of a FB_JsonDomParser [▶ 19] to the end of an array.

Syntax

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

Return value

Name	Type
PushbackCopyValue	SJsonValue

Inputs

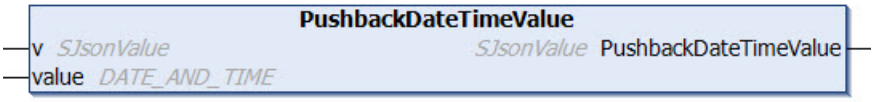
Name	Type
v	SJsonValue
json	SJsonValue

Sample call:

```
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



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

Syntax

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

Return value

Name	Type
PushbackDateTimeValue	SJsonValue

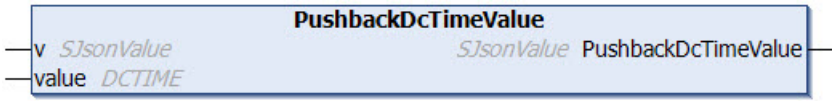
Inputs

Name	Type
v	sJsonValue
value	DATE_AND_TIME

Sample call:

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

4.1.79 PushbackDcTimeValue



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

Syntax

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

Return value

Name	Type
PushbackDcTimeValue	SJsonValue

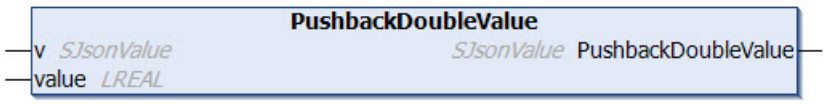
Inputs

Name	Type
v	sJsonValue
value	DCTIME

Sample call:

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

4.1.80 PushbackDoubleValue



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

Syntax

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

Return value

Name	Type
PushbackDoubleValue	SJsonValue

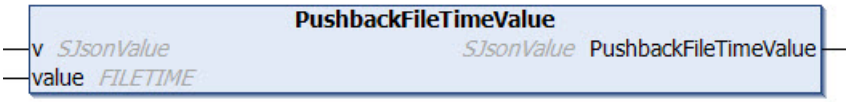
Inputs

Name	Type
v	sJsonValue
value	LREAL

Sample call:

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

4.1.81 PushbackFileTimeValue



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

Syntax

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

Return value

Name	Type
PushbackFileTimeValue	SJsonValue

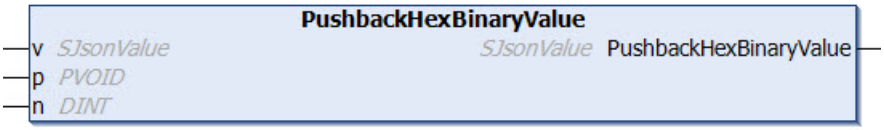
Inputs

Name	Type
v	sJsonValue
value	FILETIME

Sample call:

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

4.1.82 PushbackHexBinaryValue



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

Syntax

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

Return value

Name	Type
PushbackHexBinaryValue	SJsonValue

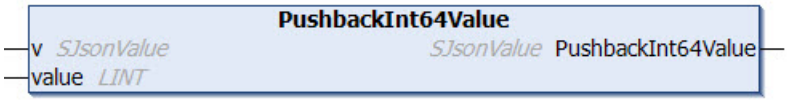
Inputs

Name	Type
v	sJsonValue
p	PVOID
n	DINT

Sample call:

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

4.1.83 PushbackInt64Value



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

Syntax

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

Return value

Name	Type
PushbackInt64Value	SJsonValue

Inputs

Name	Type
v	sJsonValue
value	LINT

Sample call:

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

4.1.84 PushbackIntValue

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

Syntax

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

Return value

Name	Type
PushbackIntValue	SJsonValue

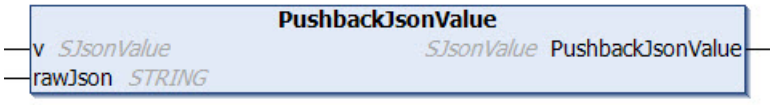
Inputs

Name	Type
v	sJsonValue
value	DINT

Sample call:

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

4.1.85 PushbackJsonValue



This method appends a JSON document that exists as a STRING in the PLC to the end of an array.

Syntax

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

Return value

Name	Type
PushbackJsonValue	SJsonValue

Inputs

Name	Type
v	SJsonValue

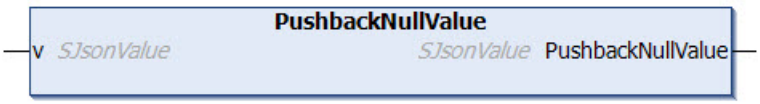
/ Inputs/Outputs

Name	Type
rawJson	STRING

Sample call:

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

4.1.86 PushbackNullValue



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

Syntax

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

Return value

Name	Type
PushbackNullValue	SJsonValue

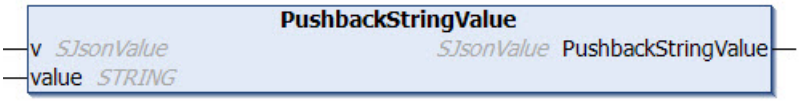
Inputs

Name	Type
v	SJsonValue

Sample call:

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

4.1.87 PushbackStringValue



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

Syntax

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

Return value

Name	Type
PushbackStringValue	SJsonValue

Inputs

Name	Type
v	SJsonValue

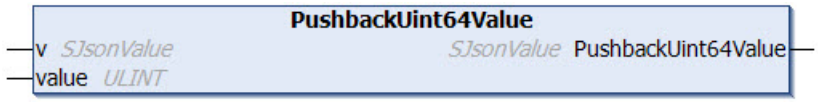
/ Inputs/Outputs

Name	Type
value	STRING

Sample call:

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

4.1.88 PushbackUint64Value



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

Syntax

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

Return value

Name	Type
PushbackUint64Value	SJsonValue

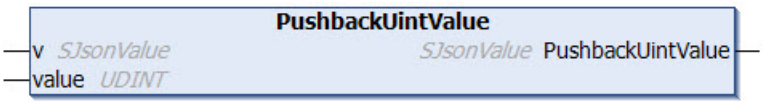
Inputs

Name	Type
v	SJsonValue
value	ULINT

Sample call:

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

4.1.89 PushbackUintValue



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

Syntax

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

Return value

Name	Type
PushbackUintValue	SJsonValue

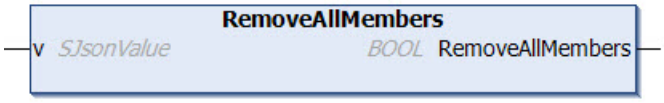
Inputs

Name	Type
v	SJsonValue
value	UDINT

Sample call:

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

4.1.90 RemoveAllMembers



This method removes all child elements from a given property.

Syntax

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

Return value

Name	Type
RemoveAllMembers	BOOL

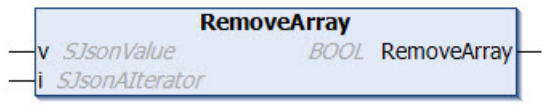
Inputs

Name	Type
v	SJsonValue

Sample call:

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

4.1.91 RemoveArray



This method deletes the value of the current array iterator.

Syntax

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

Return value

Name	Type
RemoveArray	BOOL

Inputs

Name	Type
v	SJsonValue
i	SJsonAIterator

Sample call:

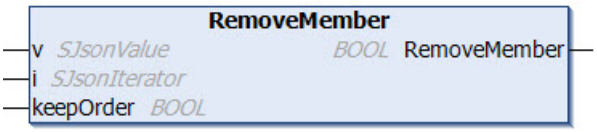
The following JSON document is to be loaded into the DOM memory:

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

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

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

4.1.92 RemoveMember



This method deletes the property at the current iterator.

Syntax

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

Return value

Name	Type
RemoveMember	BOOL

Inputs

Name	Type
v	SJsonValue
i	SJsonIterator
keepOrder	BOOL

Sample call:

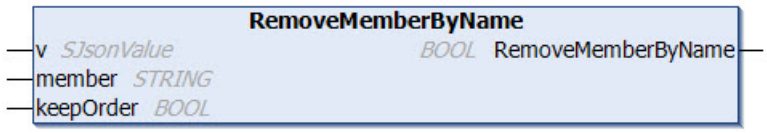
The following JSON document is to be loaded into the DOM memory:

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

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

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

4.1.93 RemoveMemberByName



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

Syntax

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

Return value

Name	Type
RemoveMemberByName	BOOL

Inputs

Name	Type
v	SJsonValue
keepOrder	BOOL

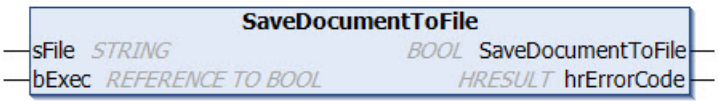
/ Inputs/Outputs

Name	Type
member	STRING

Sample call:

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

4.1.94 SaveDocumentToFile



This method saves a JSON document in a file.

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

Syntax

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

Return value

Name	Type
SaveDocumentToFile	BOOL

Inputs

Name	Type
bExec	REFERENCE TO BOOL

Inputs/Outputs

Name	Type
sFile	STRING

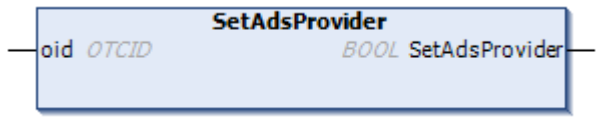
Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

4.1.95 SetAdsProvider



This method sets the context in which the file accesses are processed when loading and saving the JSON documents. If no ADS provider is specified, the current task is used.

Syntax

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

Return value

Name	Type	Description
SetAdsProvider	BOOL	The method returns TRUE if the setting of the ADS provider was successful.

Inputs

Name	Type	Description
oid	OTCID	Object ID of the task.

Sample call:

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

4.1.96 SetArray

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

Syntax

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

Return value

Name	Type
SetArray	SJsonValue

Inputs

Name	Type
v	SJsonValue

Sample call:

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

4.1.97 SetBase64

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

Syntax

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

Return value

Name	Type
SetBase64	SJsonValue

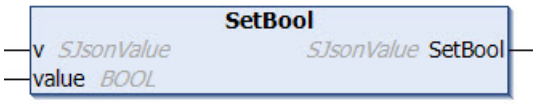
Inputs

Name	Type
v	SJsonValue
p	PVOID
n	DINT

Sample call:

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

4.1.98 SetBool



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

Syntax

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

Return value

Name	Type
SetBool	SJsonValue

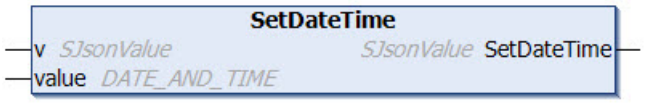
Inputs

Name	Type
v	SJsonValue
value	BOOL

Sample call:

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

4.1.99 SetDateTime



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

Syntax

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

Return value

Name	Type
SetDateTime	SJsonValue

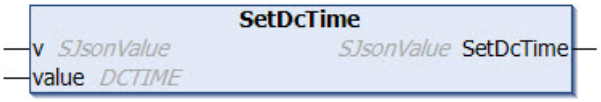
Inputs

Name	Type
v	SJsonValue
value	DATE_AND_TIME

Sample call:

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

4.1.100 SetDcTime



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

Syntax

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

Return value

Name	Type
SetDcTime	SJsonValue

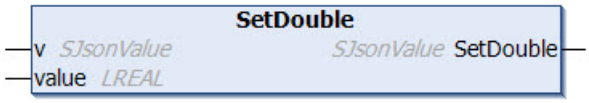
Inputs

Name	Type
v	SJsonValue
value	DCTIME

Sample call:

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

4.1.101 SetDouble



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

Syntax

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

Return value

Name	Type
SetDouble	SJsonValue

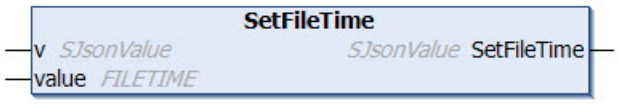
Inputs

Name	Type
v	SJsonValue
value	LREAL

Sample call:

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

4.1.102 SetFileTime



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

Syntax

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

Return value

Name	Type
SetFileTime	SJsonValue

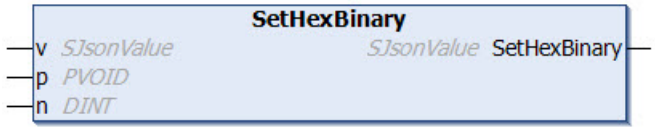
Inputs

Name	Type
v	SJsonValue
value	FILETIME

Sample call:

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

4.1.103 SetHexBinary



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

Syntax

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

Return value

Name	Type
SetHexBinary	SJsonValue

Inputs

Name	Type
v	SJsonValue
p	PVOID
n	DINT

Sample call:

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

4.1.104 SetInt

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

Syntax

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

Return value

Name	Type
SetInt	SJsonValue

Inputs

Name	Type
v	SJsonValue
value	DINT

Sample call:

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

4.1.105 SetInt64

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

Syntax

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

Return value

Name	Type
SetInt64	SJsonValue

Inputs

Name	Type
v	SJsonValue
value	LINT

Sample call:

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

4.1.106 SetJson

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

Syntax

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

Return value

Name	Type
SetJson	SJsonValue

Inputs

Name	Type
v	SJsonValue

/ Inputs/Outputs

Name	Type
rawJson	STRING

Sample call:

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

4.1.107 SetMaxDecimalPlaces

This function determines the number of decimal places after which a floating-point number is truncated.

Syntax

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

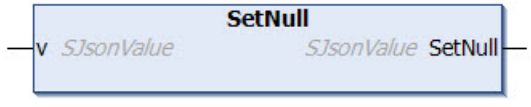
Inputs

Name	Type
dp	DINT

Sample call:

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

4.1.108 SetNull



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

Syntax

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

Return value

Name	Type
SetNull	SJsonValue

Inputs

Name	Type
v	SJsonValue

Sample call:

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

4.1.109 SetObject

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

Syntax

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

Return value

Name	Type
SetObject	SJsonValue

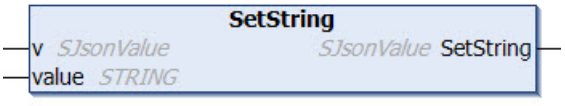
Inputs

Name	Type
v	SJsonValue

Sample call:

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

4.1.110 SetString



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

Syntax

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

Return value

Name	Type
SetString	SJsonValue

Inputs

Name	Type
v	SJsonValue

/ Inputs/Outputs

Name	Type
value	STRING

Sample call:

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

4.1.111 SetUint

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

Syntax

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

Return value

Name	Type
SetUint	SJsonValue

Inputs

Name	Type
v	SJsonValue
value	UDINT

Sample call:

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

4.1.112 SetUint64

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

Syntax

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

Return value

Name	Type
SetUint64	SJsonValue

Inputs

Name	Type
v	SJsonValue
value	ULINT

Sample call:

```
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
```

Return value

Name	Type
Swap	BOOL

Inputs

Name	Type
v	SJsonValue
w	SJsonValue

4.2 FB_JsonDynDomParser

This function block is derived from the same internal function block as FB_JsonDomParser [▶ 19] and thus offers the same interface.

The two derived function blocks differ only in their internal memory management. The FB_JsonDynDomParser is optimized for JSON documents where many changes are made and the JSON document is not released in between. It releases the allocated memory after an action has been executed, e.g. in the methods SetObject() or SetJson(). This flexibility results in greater overhead, allowing the FB_JsonDomParser [▶ 19] to improve the performance.

Strings in UTF-8 format

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

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

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

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

Syntax

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

Outputs

Name	Type
initStatus	HRESULT

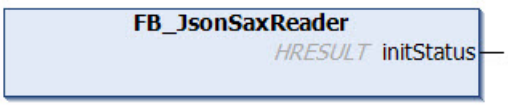
Methods

All methods can be found in FB_JsonDomParser [▶ 19].

Requirements

TwinCAT version	Hardware	Libraries to be integrated
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
```

Outputs

Name	Type
initStatus	HRESULT

Methods

Name	Description
DecodeBase64 [▶ 89]	Converts a Base64-formated string to binary data.
DecodeDateTime [▶ 90]	Creates a PLC variable of the data type DATE_AND_TIME or DT from a standardized ISO8601 time format (e.g. YYYY-MM-DDThh:mm:ss).
DecodeDcTime [▶ 91]	Creates a PLC variable of the data type DCTIME from a standardized ISO8601 time format (e.g. YYYY-MM-DDThh:mm:ss).
DecodeFileTime [▶ 91]	Creates a PLC variable of the data type FILETIME from a standardized ISO8601 time format (e.g. YYYY-MM-DDThh:mm:ss).
DecodeHexBinary [▶ 92]	Converts a string containing hexadecimal values into binary data.
IsBase64 [▶ 93]	Checks whether the transferred string corresponds to the Base64 format.
IsHexBinary [▶ 93]	Checks whether the transferred string consists of hexadecimal values.
IsISO8601TimeFormat [▶ 94]	Checks whether the transferred string corresponds to the standardized ISO8601 time format.
Parse [▶ 94]	Starts the SAX reader parsing process.
ParseValues [▶ 95]	Starts the SAX reader parsing process.

Requirements

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

4.3.1 DecodeBase64

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

Syntax

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

Return value

Name	Type
DecodeBase64	BOOL

Inputs

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

Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

4.3.2 DecodeDateTime

This method enables the generation of a PLC variable of the type DATE_AND_TIME or DT from a standardized ISO8601 time format (e.g. YYYY-MM-DDThh:mm:ss). DT corresponds to the number of seconds starting from the date 01.01.1970. If the conversion was successful the method returns TRUE, otherwise it returns FALSE.

Syntax

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

Return value

Name	Type
DecodeDateTime	BOOL

Inputs/Outputs

Name	Type
sDT	STRING

Outputs

Name	Type
nDT	DATE_AND_TIME
hrErrorCode	HRESULT

Sample call:

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

4.3.3 DecodeDcTime

This method enables the generation of a PLC variable of the type DCTIME from a standardized ISO8601 time format (e.g. YYYY-MM-DDThh:mm:ss). DCTIME corresponds to the number of nanoseconds starting from the date 01.01.2000. If the conversion was successful the method returns TRUE, otherwise it returns FALSE.

Syntax

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

Return value

Name	Type
DecodeDcTime	BOOL

Inputs/Outputs

Name	Type
sDC	STRING

Outputs

Name	Type
nDC	DCTIME
hrErrorCode	HRESULT

Sample call:

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

4.3.4 DecodeFileTime

This method enables the generation of a PLC variable of the type FILETIME from a standardized ISO8601 time format (e.g. YYYY-MM-DDThh:mm:ss). FILETIME corresponds to the number of nanoseconds starting from the date 01.01.1601 – measured in 100 nanoseconds. If the conversion was successful the method returns TRUE, otherwise it returns FALSE.

Syntax

```
METHOD DecodeDateTime : BOOL
VAR_IN_OUT CONSTANT
  sFT          : STRING;
END_VAR
VAR_OUTPUT
```

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

Return value

Name	Type
DecodeDateTime	BOOL

/ Inputs/Outputs

Name	Type
sFT	STRING

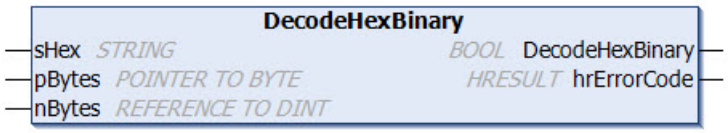
Outputs

Name	Type
nFT	FILETIME
hrErrorCode	HRESULT

Sample call:

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

4.3.5 DecodeHexBinary



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

Syntax

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

Return value

Name	Type
DecodeHexBinary	BOOL

Inputs

Name	Type
pBytes	POINTER TO BYTE
nBytes	REFERENCE TO DINT

Inputs/Outputs

Name	Type
sHex	STRING

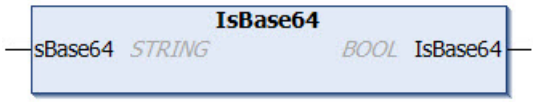
Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

4.3.6 IsBase64



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

Syntax

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

Return value

Name	Type
IsBase64	BOOL

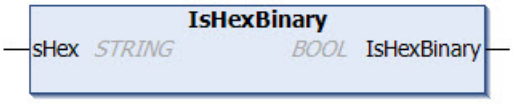
Inputs/Outputs

Name	Type
sBase64	STRING

Sample call:

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

4.3.7 IsHexBinary



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

Syntax

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

Return value

Name	Type
IsHexBinary	BOOL

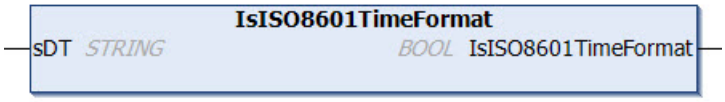
Inputs/Outputs

Name	Type
sHex	STRING

Sample call:

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

4.3.8 IsISO8601TimeFormat



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

Syntax

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

Return value

Name	Type
IsISO8601TimeFormat	BOOL

Inputs/Outputs

Name	Type
sDT	STRING

Sample call:

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

4.3.9 Parse

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

Syntax

```
METHOD Parse : BOOL
VAR_IN_OUT CONSTANT
  sJson       : STRING;
END_VAR
```

```
VAR_INPUT
  ipHdl       : ITcJsonSaxHandler;
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

Return value

Name	Type
Parse	BOOL

Inputs

Name	Type
ipHdl	ITcJsonSaxHandler

/ Inputs/Outputs

Name	Type
sJson	STRING

Outputs

Name	Type
hrErrorCode	HRESULT

4.3.10 ParseValues

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

Syntax

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

Return value

Name	Type
ParseValues	BOOL

Inputs

Name	Type
ipHdl	ITcJsonSaxValues

/ Inputs/Outputs

Name	Type
sJson	STRING

Outputs

Name	Type
hrErrorCode	HRESULT

4.4 FB_JsonSaxWriter

Strings in UTF-8 format

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

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

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

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

Syntax

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

Outputs

Name	Type
initStatus	HRESULT

Methods

Name	Description
AddBase64 [▶ 99]	Adds a value of the data type Base64 to a property.
AddBool [▶ 99]	Adds a value of the data type BOOL to a property.
AddDateTime [▶ 100]	Adds a value of the data type DATE_AND_TIME to a property.
AddDcTime [▶ 100]	Adds a value of the data type DCTIME to a property.
AddDint [▶ 100]	Adds a value of the data type DINT to a property.
AddFileTime [▶ 101]	Adds a value of the data type FILETIME to a property.
AddHexBinary [▶ 101]	Adds a HexBinary value to a property.
AddKey [▶ 102]	Adds a new property key at the current position of the SAX Writer.
AddKeyBool [▶ 102]	Creates a new property key and at the same time a value of the data type BOOL.
AddKeyDateTime [▶ 103]	Creates a new property key and at the same time a value of the data type DATE_AND_TIME.
AddKeyDcTime [▶ 103]	Creates a new property key and at the same time a value of the data type DCTIME.
AddKeyFileTime [▶ 104]	Creates a new property key and at the same time a value of the data type FILETIME.
AddKeyLreal [▶ 104]	Creates a new property key and at the same time a value of the data type LREAL.
AddKeyNull [▶ 105]	Creates a new property key and initializes its value with null.
AddKeyNumber [▶ 105]	Creates a new property key and at the same time a value of the data type DINT.
AddKeyString [▶ 106]	Creates a new property key and at the same time a value of the data type STRING.
AddLint [▶ 106]	Adds a value of the data type LINT to a property.
AddLreal [▶ 107]	Adds a value of the data type LREAL to a property.
AddNull [▶ 107]	Adds the value null to a property.
AddRawArray [▶ 107]	Adds a valid JSON array to a given property as a value.
AddRawObject [▶ 108]	Adds a valid JSON object to a given property as a value.
AddReal [▶ 108]	Adds a value of the data type REAL to a property.
AddString [▶ 109]	Adds a value of the data type STRING to a property.
AddUdint [▶ 109]	Adds a value of the data type UDINT to a property.
AddUlint [▶ 110]	Adds a value of the data type ULINT to a property.
CopyDocument [▶ 110]	Copies the contents of the JSON object currently created with the SAX Writer into a target variable of the data type STRING.
EndArray [▶ 111]	Creates the end of a started JSON array and inserts it at the current position of the SAX Writer.
EndObject [▶ 111]	Creates the end of a started JSON object and inserts it at the current position of the SAX Writer.
GetDocument [▶ 112]	Returns the content of the JSON object that is currently created with the SAX Writer and returns it as data type STRING(255).
GetDocumentLength [▶ 112]	Returns the length of the JSON object that is currently created with the SAX Writer and returns it as data type UDINT.
GetMaxDecimalPlaces [▶ 113]	
IsComplete	
ResetDocument [▶ 113]	Resets the JSON object currently created with the SAX Writer.
SetMaxDecimalPlaces [▶ 113]	

Name	Description
StartArray [▶ 114]	Creates the start of a new JSON array and inserts it at the current position of the SAX Writer.
StartObject [▶ 114]	Creates the start of a new JSON object and inserts it at the current position of the SAX Writer.

Requirements

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

4.4.1 AddBase64

This method adds a value of the data type Base64 to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 102].

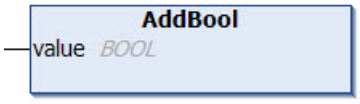
Syntax

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

Inputs

Name	Type
pBytes	POINTER TO BYTE
nBytes	DINT

4.4.2 AddBool



This method adds a value of the data type BOOL to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 102].

Syntax

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

Inputs

Name	Type
value	BOOL

Sample call:

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

4.4.3 AddDateTime

This method adds a value of the data type DATE_AND_TIME to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 102].

Syntax

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

Inputs

Name	Type
value	DATE_AND_TIME

Sample call:

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

4.4.4 AddDcTime

This method adds a value of the data type DCTIME to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 102].

Syntax

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

Inputs

Name	Type
value	DCTIME

Sample call:

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

4.4.5 AddDint

This method adds a value of the data type DINT to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 102].

Syntax

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

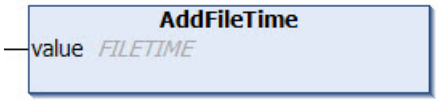
Inputs

Name	Type
value	DINT

Sample call:

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

4.4.6 AddFileTime



This method adds a value of the data type FILETIME to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 102].

Syntax

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

Inputs

Name	Type
value	FILETIME

Sample call:

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

4.4.7 AddHexBinary

This method adds a hex binary value to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 102].

Syntax

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

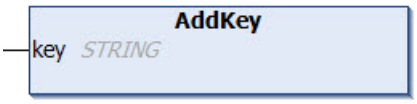
Inputs

Name	Type
pBytes	POINTER TO BYTE
nBytes	DINT

Sample call:

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

4.4.8 AddKey



This method adds a new property key at the current position of the SAX writer. The value of the new property is usually set afterwards. This can be done using one of the following methods, for example: AddBase64 [▶ 99], AddBool [▶ 99], AddDateTime [▶ 100], AddDcTime [▶ 100], AddDint [▶ 100], AddFileTime [▶ 101], AddHexBinary [▶ 101], AddLint [▶ 106], AddLreal [▶ 107], AddNull [▶ 107], AddRawArray [▶ 107], AddRawObject [▶ 108], AddReal [▶ 108], AddString [▶ 109], AddUdint [▶ 109], AddUlint [▶ 110].

Syntax

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

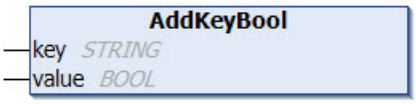
/ Inputs/Outputs

Name	Type
key	STRING

Sample call:

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

4.4.9 AddKeyBool



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

Syntax

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

Inputs

Name	Type
value	BOOL

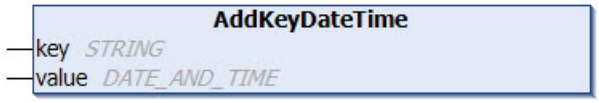
/ Inputs/Outputs

Name	Type
key	STRING

Sample call:

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

4.4.10 AddKeyDateTime



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

Syntax

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

Inputs

Name	Type
value	DATE_AND_TIME

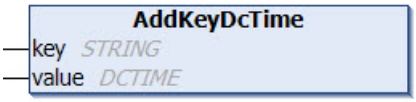
/ Inputs/Outputs

Name	Type
key	STRING

Sample call:

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

4.4.11 AddKeyDcTime



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

Syntax

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

Inputs

Name	Type
value	DCTIME

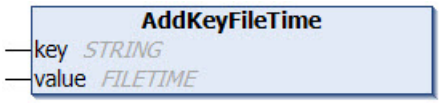
Inputs/Outputs

Name	Type
key	STRING

Sample call:

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

4.4.12 AddKeyFileTime



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

Syntax

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

Inputs

Name	Type
value	FILETIME

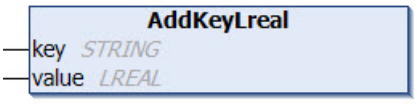
Inputs/Outputs

Name	Type
key	STRING

Sample call:

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

4.4.13 AddKeyLreal



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

Syntax

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

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

Inputs

Name	Type
value	LREAL

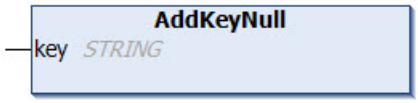
/ Inputs/Outputs

Name	Type
key	STRING

Sample call:

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

4.4.14 AddKeyNull



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

Syntax

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

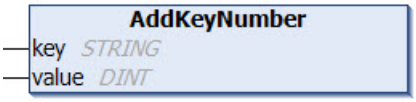
/ Inputs/Outputs

Name	Type
key	STRING

Sample call:

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

4.4.15 AddKeyNumber



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

Syntax

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

Inputs

Name	Type
value	DINT

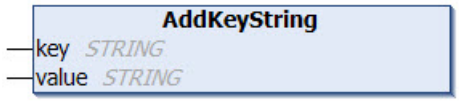
/ Inputs/Outputs

Name	Type
key	STRING

Sample call:

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

4.4.16 AddKeyString



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

Syntax

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

/ Inputs/Outputs

Name	Type
key	STRING
value	STRING

Sample call:

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

4.4.17 AddLint

This method adds a value of the data type LINT to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 102].

Syntax

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

Inputs

Name	Type
value	LINT

Sample call:

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

4.4.18 AddLreal

This method adds a value of the data type LREAL to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 102].

Syntax

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

Inputs

Name	Type
value	LREAL

Sample call:

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

4.4.19 AddNull

This method adds the value null to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 102].

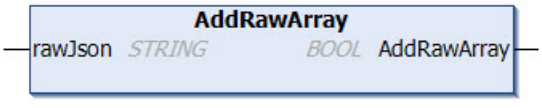
Syntax

```
METHOD AddNull
```

Sample call:

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

4.4.20 AddRawArray



This method adds a valid JSON array to a given property as a value. The array to be added must be in a valid JSON format and may only be added if the SAX Writer is at a correspondingly valid position, e.g. directly after a preceding AddKey() [▶ 102], StartArray() [▶ 114] or as the first call after a ResetDocument() [▶ 113].

Syntax

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

Return value

Name	Type
AddRawArray	BOOL

/ Inputs/Outputs

Name	Type
rawJson	STRING

Sample call:

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

4.4.21 AddRawObject

This method adds a valid JSON object to a given property as a value. The object to be added must be in a valid JSON format and may only be added if the SAX Writer is in an appropriately valid position, e.g. directly after a preceding AddKey() [▶ 102], StartArray() [▶ 114] or as the first call after a ResetDocument() [▶ 113].

Syntax

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

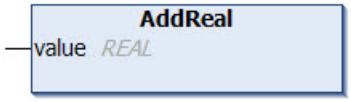
/ Inputs/Outputs

Name	Type
rawJson	STRING

Sample call:

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

4.4.22 AddReal



This method adds a value of the data type REAL to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 102].

Syntax

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

Inputs

Name	Type
value	REAL

Sample call:

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

4.4.23 AddString

This method adds a value of the data type STRING to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 102].

Syntax

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

/ Inputs/Outputs

Name	Type
value	STRING

Sample call:

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

4.4.24 AddUdint

This method adds a value of the data type UDINT to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 102].

Syntax

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

Inputs

Name	Type
value	UDINT

Sample call:

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

4.4.25 AddUlint

This method adds a value of the data type ULINT to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 102].

Syntax

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

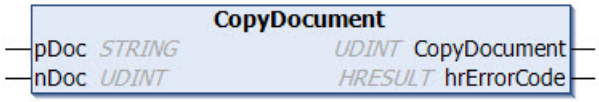
Inputs

Name	Type
value	ULINT

Sample call:

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

4.4.26 CopyDocument



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

Syntax

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

Return value

Name	Type
CopyDocument	UDINT

Inputs

Name	Type
nDoc	UDINT

/ Inputs/Outputs

Name	Type
pDoc	STRING

Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

4.4.27 EndArray

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

Syntax

```
METHOD EndArray : HRESULT
```

Return value

Name	Type
EndArray	HRESULT

Sample call:

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

4.4.28 EndObject

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

Syntax

```
METHOD EndObject : HRESULT
```

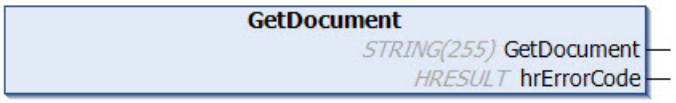
Return value

Name	Type
EndObject	HRESULT

Sample call:

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

4.4.29 GetDocument



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

The maximum size of the string returned by the method is 255 characters. With longer strings, the method will return a NULL string. In this case the method CopyDocument [▶ 110]() must be used.

Syntax

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

Return value

Name	Type
GetDocument	STRING(255)

Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

4.4.30 GetDocumentLength

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

Syntax

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

Return value

Name	Type
GetDocumentLength	UDINT

Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

4.4.31 GetMaxDecimalPlaces

This function queries the currently set number of decimal places after which a floating point number is truncated.

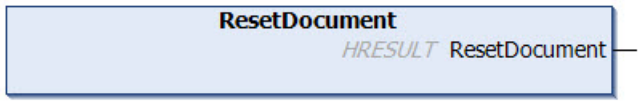
Syntax

```
METHOD GetMaxDecimalPlaces : DINT
```

Return value

Name	Type	Description
GetMaxDecimalPlaces	DINT	The currently set number of digits after which a floating point number is truncated.

4.4.32 ResetDocument



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

Syntax

```
METHOD ResetDocument : HRESULT
```

Return value

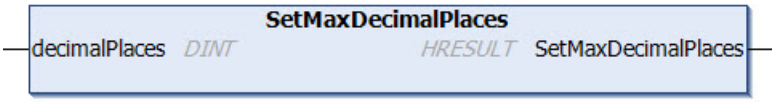
Name	Type
ResetDocument	HRESULT

Sample call:

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

4.4.33 SetMaxDecimalPlaces

This function determines the number of decimal places after which a floating-point number is truncated.



Syntax

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

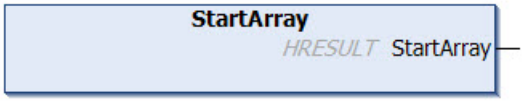
Return value

Name	Type
SetMaxDecimalPlaces	HRESULT

Inputs

Name	Type	Description
decimalPlaces	DINT	The number of decimal places to be set after which a floating-point number is truncated.

4.4.34 StartArray



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

Syntax

```
METHOD StartArray : HRESULT
```

Return value

Name	Type
StartArray	HRESULT

Sample call:

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

4.4.35 StartObject

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

Syntax

```
METHOD StartObject : HRESULT
```

Return value

Name	Type
StartObject	HRESULT

Sample call:

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

4.5 FB_JsonSaxPrettyWriter

This function block has a difference from FB_JsonSaxWriter [▶ 96]: it adds matching whitespaces for better readability of a JSON document.

Strings in UTF-8 format

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

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

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

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

Syntax

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

Outputs

Name	Type
initStatus	HRESULT

Methods

Name	Description
SetFormatOptions [▶ 130]	Adjusts the general formatting options for an instance of the [<=>] FB_JsonSaxPrettyWriter.
SetIndent [▶ 130]	Customizes the indentation.

All other methods can be found at FB_JsonSaxWriter [▶ 96].

Requirements

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

4.5.1 AddBase64

This method adds a value of the data type Base64 to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 118].

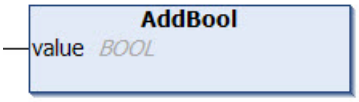
Syntax

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

Inputs

Name	Type
pBytes	POINTER TO BYTE
nBytes	DINT

4.5.2 AddBool



This method adds a value of the data type BOOL to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 118].

Syntax

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

Inputs

Name	Type
value	BOOL

Sample call:

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

4.5.3 AddDateTime

This method adds a value of the data type DATE_AND_TIME to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 118].

Syntax

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

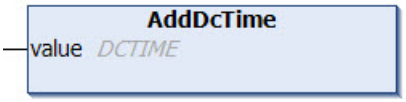
Inputs

Name	Type
value	DATE_AND_TIME

Sample call:

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

4.5.4 AddDcTime



This method adds a value of the data type DCTIME to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 118].

Syntax

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

Inputs

Name	Type
value	DCTIME

Sample call:

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

4.5.5 AddDint

This method adds a value of the data type DINT to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 118].

Syntax

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

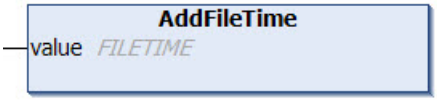
Inputs

Name	Type
value	DINT

Sample call:

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

4.5.6 AddFileTime



This method adds a value of the data type FILETIME to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 118].

Syntax

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

Inputs

Name	Type
value	FILETIME

Sample call:

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

4.5.7 AddHexBinary

This method adds a hex binary value to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 118].

Syntax

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

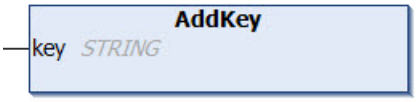
Inputs

Name	Type
pBytes	POINTER TO BYTE
nBytes	DINT

Sample call:

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

4.5.8 AddKey



This method adds a new property key at the current position of the SAX writer. The value of the new property is usually set afterwards. This can be done using one of the following methods, for example: AddBase64 [▶ 115], AddBool [▶ 116], AddDateTime [▶ 116], AddDcTime [▶ 117], AddDint [▶ 117], AddFileTime [▶ 117], AddHexBinary [▶ 118], AddLint [▶ 123], AddLreal [▶ 123], AddNull [▶ 123], AddRawArray [▶ 124], AddRawObject [▶ 124], AddReal [▶ 125], AddString [▶ 125], AddUdint [▶ 126], AddUlint [▶ 126].

Syntax

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

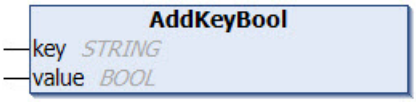
/ Inputs/Outputs

Name	Type
key	STRING

Sample call:

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

4.5.9 AddKeyBool



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

Syntax

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

Inputs

Name	Type
value	BOOL

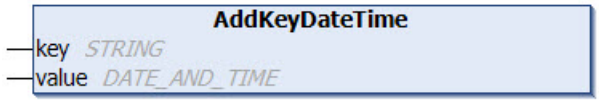
/ Inputs/Outputs

Name	Type
key	STRING

Sample call:

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

4.5.10 AddKeyDateTime



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

Syntax

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

Inputs

Name	Type
value	DATE_AND_TIME

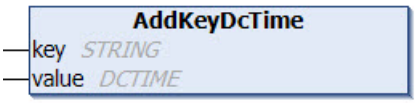
/ Inputs/Outputs

Name	Type
key	STRING

Sample call:

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

4.5.11 AddKeyDcTime



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

Syntax

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

Inputs

Name	Type
value	DCTIME

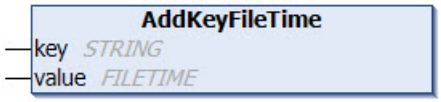
/ Inputs/Outputs

Name	Type
key	STRING

Sample call:

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

4.5.12 AddKeyFileTime



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

Syntax

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

Inputs

Name	Type
value	FILETIME

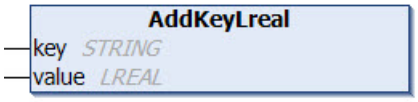
/ Inputs/Outputs

Name	Type
key	STRING

Sample call:

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

4.5.13 AddKeyLreal



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

Syntax

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

Inputs

Name	Type
value	LREAL

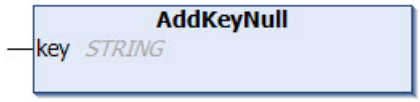
/ Inputs/Outputs

Name	Type
key	STRING

Sample call:

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

4.5.14 AddKeyNull



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

Syntax

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

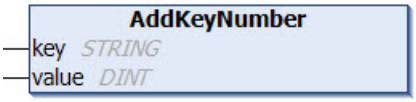
/ Inputs/Outputs

Name	Type
key	STRING

Sample call:

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

4.5.15 AddKeyNumber



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

Syntax

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

Inputs

Name	Type
value	DINT

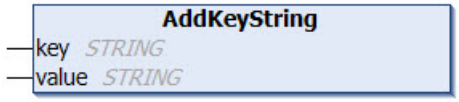
/ Inputs/Outputs

Name	Type
key	STRING

Sample call:

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

4.5.16 AddKeyString



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

Syntax

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

/ Inputs/Outputs

Name	Type
key	STRING
value	STRING

Sample call:

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

4.5.17 AddLint

This method adds a value of the data type LINT to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 118].

Syntax

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

Inputs

Name	Type
value	LINT

Sample call:

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

4.5.18 AddLreal

This method adds a value of the data type LREAL to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 118].

Syntax

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

Inputs

Name	Type
value	LREAL

Sample call:

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

4.5.19 AddNull

This method adds the value null to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 118].

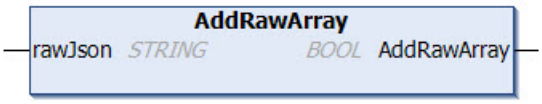
Syntax

```
METHOD AddNull
```

Sample call:

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

4.5.20 AddRawArray



This method adds a valid JSON array to a given property as a value. The array to be added must be in a valid JSON format and may only be added if the SAX Writer is at a correspondingly valid position, e.g. directly after a preceding AddKey() [▶ 118], StartArray() [▶ 131] or as the first call after a ResetDocument() [▶ 129].

Syntax

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

Return value

Name	Type
AddRawArray	BOOL

/ Inputs/Outputs

Name	Type
rawJson	STRING

Sample call:

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

4.5.21 AddRawObject

This method adds a valid JSON object to a given property as a value. The object to be added must be in a valid JSON format and may only be added if the SAX Writer is in an appropriately valid position, e.g. directly after a preceding AddKey() [▶ 118], StartArray() [▶ 131] or as the first call after a ResetDocument() [▶ 129].

Syntax

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

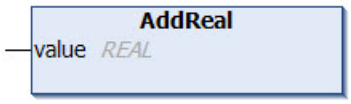
/ Inputs/Outputs

Name	Type
rawJson	STRING

Sample call:

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

4.5.22 AddReal



This method adds a value of the data type REAL to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 118].

Syntax

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

Inputs

Name	Type
value	REAL

Sample call:

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

4.5.23 AddString

This method adds a value of the data type STRING to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 118].

Syntax

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

/ Inputs/Outputs

Name	Type
value	STRING

Sample call:

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

4.5.24 AddUdint

This method adds a value of the data type UDINT to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 118].

Syntax

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

Inputs

Name	Type
value	UDINT

Sample call:

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

4.5.25 AddUlint

This method adds a value of the data type ULINT to a property. Usually, a corresponding property was created beforehand with the method AddKey() [▶ 118].

Syntax

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

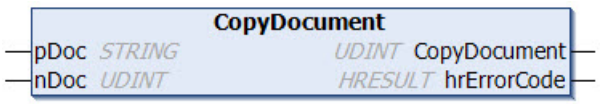
Inputs

Name	Type
value	ULINT

Sample call:

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

4.5.26 CopyDocument



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

Syntax

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

Return value

Name	Type
CopyDocument	UDINT

Inputs

Name	Type
nDoc	UDINT

Inputs/Outputs

Name	Type
pDoc	STRING

Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

4.5.27 EndArray

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

Syntax

```
METHOD EndArray : HRESULT
```

Return value

Name	Type
EndArray	HRESULT

Sample call:

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

4.5.28 EndObject

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

Syntax

```
METHOD EndObject : HRESULT
```

Return value

Name	Type
EndObject	HRESULT

Sample call:

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

4.5.29 GetDocument

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

The maximum size of the string returned by the method is 255 characters. With longer strings, the method will return a NULL string. In this case the method CopyDocument [▶ 126]() must be used.

Syntax

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

Return value

Name	Type
GetDocument	STRING(255)

Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

4.5.30 GetDocumentLength

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

Syntax

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

Return value

Name	Type
GetDocumentLength	UDINT

Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

4.5.31 GetMaxDecimalPlaces

This function queries the currently set number of decimal places after which a floating point number is truncated.

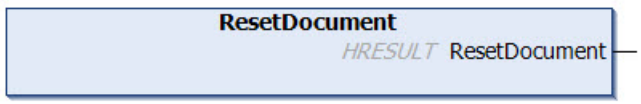
Syntax

```
METHOD GetMaxDecimalPlaces : DINT
```

Return value

Name	Type	Description
GetMaxDecimalPlaces	DINT	The currently set number of digits after which a floating point number is truncated.

4.5.32 ResetDocument



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

Syntax

```
METHOD ResetDocument : HRESULT
```

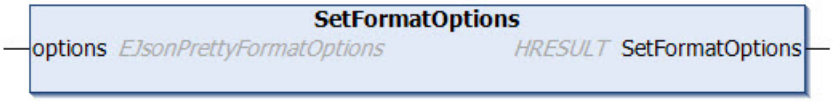
Return value

Name	Type
ResetDocument	HRESULT

Sample call:

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

4.5.33 SetFormatOptions



This method is used to customize the general formatting options for an instance of the FB_JsonSaxPrettyWriter [▶ 114]. In addition to the default configuration, there is also the possibility that no line breaks are generated within an array. The array is then output in one line despite the inserted indentations.

Syntax

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

Return value

Name	Type
SetFormatOptions	HRESULT

Inputs

Name	Type
options	EJsonPrettyFormatOptions

Sample call:

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

4.5.34 SetIndent

This method is used to customize the indentation.

Syntax

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

Return value

Name	Type
SetIndent	HRESULT

Inputs

Name	Type
indentChar	SINT
indentCharCount	UDINT

Sample call:

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

4.5.35 SetMaxDecimalPlaces

This function determines the number of decimal places after which a floating-point number is truncated.

Syntax

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

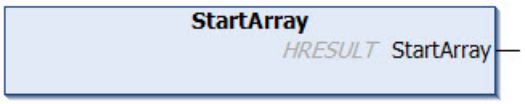
Return value

Name	Type
SetMaxDecimalPlaces	HRESULT

Inputs

Name	Type	Description
decimalPlaces	DINT	The number of decimal places to be set after which a floating-point number is truncated.

4.5.36 StartArray



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

Syntax

```
METHOD StartArray : HRESULT
```

Return value

Name	Type
StartArray	HRESULT

Sample call:

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

4.5.37 StartObject

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

Syntax

```
METHOD StartObject : HRESULT
```

Return value

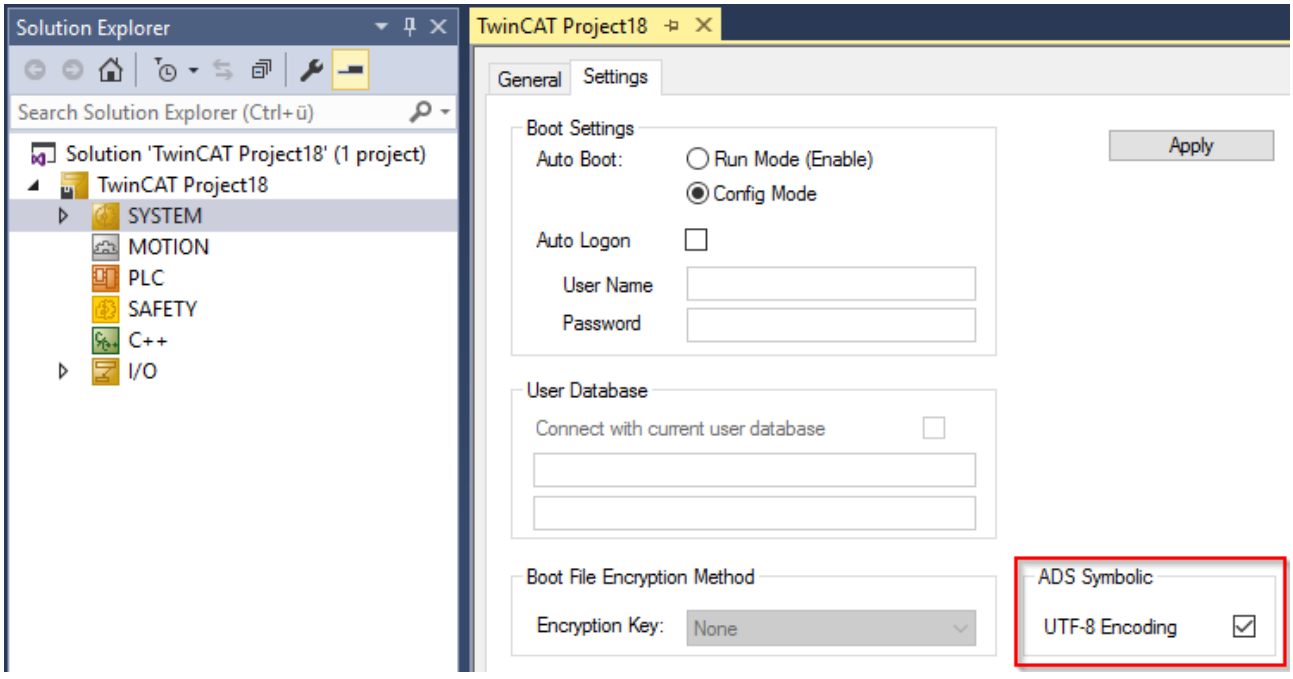
Name	Type
StartObject	HRESULT

Sample call:

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

4.6 FB_JsonReadWriteDataType

In order to use UTF-8 characters, e.g. in the automatic generation of metadata via the function block FB_JsonReadWriteDataType [▶ 132], the check box for the support of UTF-8 in the symbolism must be activated in the TwinCAT project. To do this, double-click on **SYSTEM** in the project tree, open the **Settings** tab and activate the corresponding check box.



Strings in UTF-8 format

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

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

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

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

Outputs

Name	Type
initStatus	HRESULT

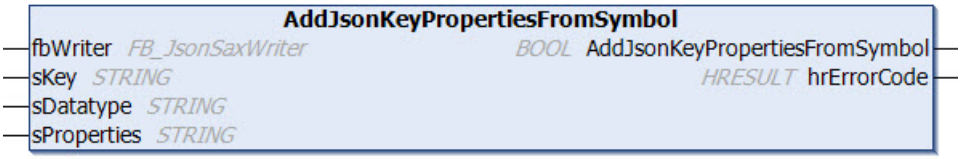
Methods

Name	Description
FB_JsonReadWriteDataType	Metadata can be added via PLC attributes to the JSON representation of a PLC data structure on an FB_JsonSaxWriter object.
AddJsonKeyValueFromSymbol [▶ 135]	Creates the JSON representation of a PLC data structure on an FB_JsonSaxWriter object.
AddJsonValueFromSymbol [▶ 136]	Creates the JSON representation of a PLC data structure on an FB_JsonSaxWriter object.
CopyJsonStringFromSymbol [▶ 137]	Creates the JSON representation of a symbol and copies it to a variable of data type STRING.
CopyJsonStringFromSymbolProperties [▶ 138]	Creates a corresponding JSON representation of PLC attributes on a symbol.
CopySymbolNameByAddress [▶ 139]	Returns the complete (ADS) symbol name of a transferred symbol.
GetDataTypeNameByAddress [▶ 140]	Returns the data type name of a transferred symbol.
GetJsonFromSymbol [▶ 140]	Creates the corresponding JSON representation of a symbol.
GetJsonStringFromSymbol [▶ 141]	Creates the corresponding JSON representation of a symbol.
GetJsonStringFromSymbolProperties [▶ 142]	Creates a corresponding JSON representation of PLC attributes on a symbol.
GetSizeJsonStringFromSymbol [▶ 143]	Reads the size of the JSON representation of a symbol.
GetSizeJsonStringFromSymbolProperties [▶ 144]	Reads the size of the JSON representation of PLC attributes on a symbol.
GetSymbolNameByAddress [▶ 144]	Returns the complete (ADS) symbol name of a transferred symbol.
SetSymbolFromJson [▶ 145]	Extracts a string containing a valid JSON message.

Requirements

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

4.6.1 AddJsonKeyPropertiesFromSymbol



With the aid of this method, metadata can be added via PLC attributes to the JSON representation of a PLC data structure on an FB_JsonSaxWriter [▶ 96] object. The method receives as its input parameters the instance of the FB_JsonSaxWriter function block, the desired name of the JSON property that is to contain the metadata, the data type name of the structure and a string variable sProperties, which contains a list of the PLC attributes to be extracted, separated by a cross bar.

Syntax

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

Return value

Name	Type
AddJsonValueFromSymbol	BOOL

Inputs/Outputs

Name	Type
fbWriter	FB_JsonSaxWriter
sKey	STRING
sDatatype	STRING
sProperties	STRING

Outputs

Name	Type
hrErrorCode	HRESULT

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

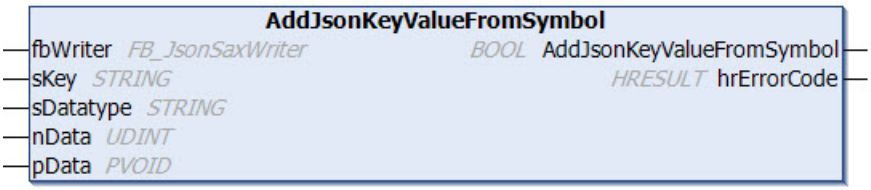
```
{attribute 'Unit'        := 'm/s'}
{attribute 'DisplayName' := 'Speed'}
Sensor1                  : REAL;
```

A complete sample of how to use this method can be found in section Tc3JsonXmlSampleJsonDataType [▶ 216].

Sample call:

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

4.6.2 AddJsonKeyValueFromSymbol



This method generates the JSON representation of a PLC data structure on an FB_JsonSaxWriter [▶ 96] object. The method receives as its input parameters the instance of the FB_JsonSaxWriter function block, the data type name of the structure, and the address and size of the source structure instance. As a result, the FB_JsonSaxWriter instance contains a valid JSON representation of the structure. Unlike the method AddJsonValueFromSymbol() [▶ 136], the elements of the source structure are nested here in a JSON sub-object whose name can be specified via the input/output parameter sKey.

Syntax

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

Return value

Name	Type
AddJsonValueFromSymbol	BOOL

Inputs

Name	Type
nData	UDINT
pData	PVOID

/ Inputs/Outputs

Name	Type
fbWriter	FB_JsonSaxWriter
sKey	STRING
sDatatype	STRING

Outputs

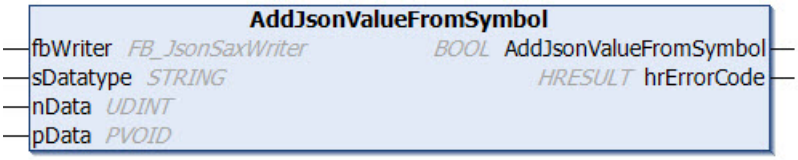
Name	Type
hrErrorCode	HRESULT

A complete sample of how to use this method can be found in section Tc3JsonXmlSampleJsonDataType [▶ 216].

Sample call:

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

4.6.3 AddJsonValueFromSymbol



This method generates the JSON representation of a PLC data structure on an FB_JsonSaxWriter [▶ 96] object. The method receives as its input parameters the instance of the FB_JsonSaxWriter function block, the data type name of the structure, and the address and size of the source structure instance. As a result, the FB_JsonSaxWriter instance contains a valid JSON representation of the structure.

Syntax

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

Return value

Name	Type
AddJsonValueFromSymbol	BOOL

Inputs

Name	Type
nData	UDINT
pData	PVOID

/ Inputs/Outputs

Name	Type
fbWriter	FB_JsonSaxWriter
sDatatype	STRING

Outputs

Name	Type
hrErrorCode	HRESULT

A complete sample of how to use this method can be found in section Tc3JsonXmlSampleJsonDataType [▶ 216].

Sample call:

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

4.6.4 CopyJsonStringFromSymbol

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

Syntax

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

Return value

Name	Type
CopyJsonStringFromSymbol	UDINT

Inputs

Name	Type
nData	UDINT
nDoc	UDINT
pData	PVOID

Inputs/Outputs

Name	Type
pDoc	STRING
sDatatype	STRING

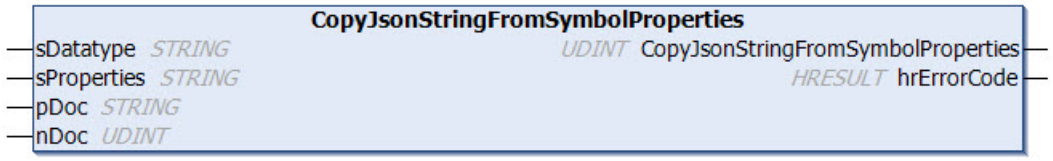
Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

4.6.5 CopyJsonStringFromSymbolProperties



This method generates a corresponding JSON representation of PLC attributes on a symbol. In contrast to the AddJsonKeyPropertiesFromSymbol [▶ 134] method, the result is not written to an instance of the function block FB_JsonSaxWriter, but to a string variable. The method receives as its input parameters the data type name of the symbol and a string variable that represents a list of the PLC attributes to be extracted, separated by a cross bar.

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

Syntax

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

Return value

Name	Type
CopyJsonStringFromSymbolProperties	UDINT

Inputs

Name	Type
nDoc	UDINT

/ Inputs/Outputs

Name	Type
pDoc	STRING
sDatatype	STRING
sProperties	STRING

Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

4.6.6 CopySymbolNameByAddress

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

Syntax

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

Return value

Name	Type
CopySymbolNameByAddress	UDINT

Inputs

Name	Type
nData	UDINT
pData	PVOID
nName	UDINT

/ Inputs/Outputs

Name	Type
sName	STRING

Outputs

Name	Type
hrErrorCode	HRESULT

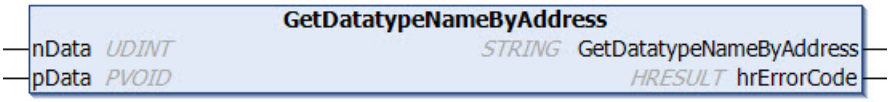
Sample call:

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

Requirements

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

4.6.7 GetDataTypeNameByAddress



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

Syntax

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

Return value

Name	Type
GetDataTypeNameByAddress	STRING

Inputs

Name	Type
nData	UDINT
pData	PVOID

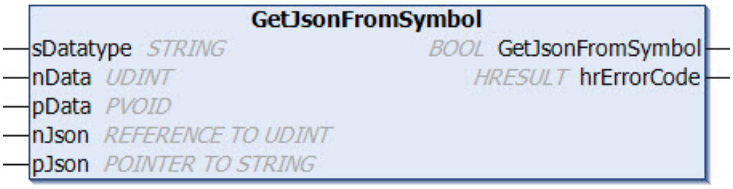
Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

4.6.8 GetJsonFromSymbol



This method generates the corresponding JSON representation of a symbol. In contrast to the AddJsonValueFromSymbol() [▶ 136] method, the result is not written to an instance of the function block FB_JsonSaxWriter, but to a string variable. The method receives as its input parameters the data type name of the symbol as well as the address and size of the source symbol, e.g. of a structure instance. The address and size of the destination buffer that contains the JSON representation of the symbol after the call are transferred as further input parameters.

Syntax

```
METHOD GetJsonFromSymbol : BOOL
VAR_IN_OUT CONSTANT
  sDatatype : STRING;
END_VAR
```

```
VAR_INPUT
  nData       : UDINT;
  pData       : PVOID;
  nJson       : REFERENCE TO UDINT;
  pJson       : POINTER TO STRING;
END_VAR
VAR_OUTPUT
  hrErrorCode : HRESULT;
END_VAR
```

Return value

Name	Type
GetJsonFromSymbol	BOOL

Inputs

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

Inputs/Outputs

Name	Type
sDatatype	STRING

Outputs

Name	Type
hrErrorCode	HRESULT

Input parameter nJson

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

Sample call:

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

4.6.9 GetJsonStringFromSymbol

This method generates the corresponding JSON representation of a symbol. In contrast to the AddJsonValueFromSymbol() [▶ 136] method, the result is not written to an instance of the function block FB_JsonSaxWriter, but to a string variable. The method receives as its input parameters the data type name of the symbol as well as the address and size of the source symbol of a structure instance, for example.

The maximum size of the string returned by the method is 255 characters. With longer strings, the method will return a NULL string. In this case the method CopyJsonStringFromSymbol [▶ 137]() must be used.

Syntax

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

Return value

Name	Type
GetJsonStringFromSymbol	STRING(255)

Inputs

Name	Type
nData	UDINT
pData	PVOID

/ Inputs/Outputs

Name	Type
sDatatype	STRING

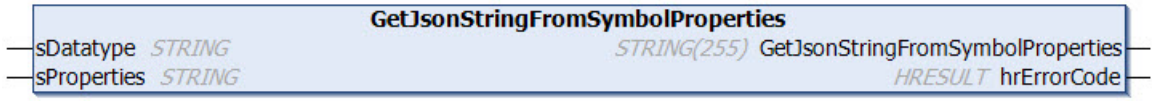
Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

4.6.10 GetJsonStringFromSymbolProperties



This method generates a corresponding JSON representation of PLC attributes on a symbol. In contrast to the AddJsonKeyPropertiesFromSymbol [▶ 134] method, the result is not written to an instance of the function block FB_JsonSaxWriter, but to a string variable. The method receives as its input parameters the data type name of the symbol and a string variable that represents a list of the PLC attributes to be extracted, separated by a cross bar. The result is returned directly as the return value of the method.

The maximum size of the string returned by the method is 255 characters. With longer strings, the method will return a NULL string. In this case the method CopyJsonStringFromSymbolProperties [▶ 138]() must be used.

Syntax

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

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

Return value

Name	Type
GetJsonStringFromSymbolProperties	STRING(255)

/ Inputs/Outputs

Name	Type
sDatatype	STRING
sProperties	STRING

Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

4.6.11 GetSizeJsonStringFromSymbol

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

Syntax

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

Return value

Name	Type
GetSizeJsonStringFromSymbol	UDINT

Inputs

Name	Type
nData	UDINT
pData	PVOID

Inputs/Outputs

Name	Type
sDatatype	STRING

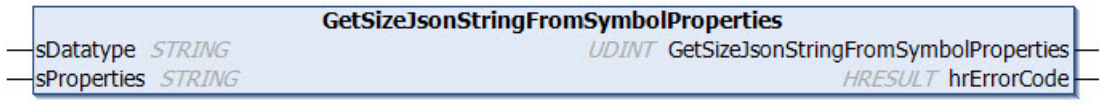
Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

4.6.12 GetSizeJsonStringFromSymbolProperties



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

Syntax

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

Return value

Name	Type
GetSizeJsonStringFromSymbolProperties	UDINT

Inputs/Outputs

Name	Type
sDatatype	STRING
sProperties	STRING

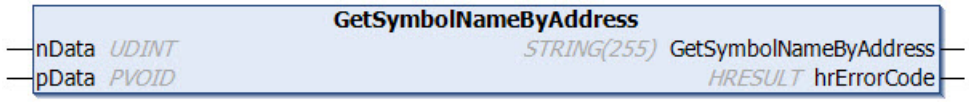
Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

4.6.13 GetSymbolNameByAddress



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

The maximum size of the string returned by the method is 255 characters. With longer strings, the method will return a null string. In this case the method CopySymbolNameByAddress() [▶ 139] must be used.

Syntax

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

Return value

Name	Type
GetSymbolNameByAddress	STRING(255)

Inputs

Name	Type
nData	UDINT
pData	PVOID

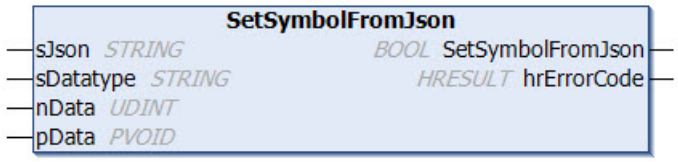
Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

4.6.14 SetSymbolFromJson



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

Syntax

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

Return value

Name	Type
SetSymbolFromJson	BOOL

Inputs

Name	Type
nData	UDINT
pData	PVOID

Inputs/Outputs

Name	Type
sJson	STRING
sDatatype	STRING

Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

4.7 FB_XmlDomParser

Strings in UTF-8 format

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

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

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

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

Outputs

Name	Type
initStatus	HRESULT

Methods

Name	Description
AppendAttribute [▶ 152]	Appends a new attribute to an existing node.
AppendAttributeAsBool [▶ 153]	Appends a new attribute to an existing node (Boolean).
AppendAttributeAsDouble [▶ 154]	Appends a new attribute to an existing node (Double).
AppendAttributeAsFloat [▶ 155]	Appends a new attribute to an existing node (Float).
AppendAttributeAsInt [▶ 155]	Appends a new attribute to an existing node (Integer).
AppendAttributeAsLint [▶ 156]	Appends a new attribute to an existing node (Integer64).
AppendAttributeAsUint [▶ 157]	Appends a new attribute to an existing node (Unsigned Integer).
AppendAttributeAsUlint [▶ 157]	Appends a new attribute to an existing node (Unsigned Integer64).
AppendAttributeCopy [▶ 158]	Appends a new attribute to an existing node.
AppendChild [▶ 159]	Appends a new node below an existing node.
AppendChildAsBool [▶ 160]	Appends a new node below an existing node (Boolean).
AppendChildAsDouble [▶ 160]	Appends a new node below an existing node (Double).
AppendChildAsFloat [▶ 161]	Appends a new node below an existing node (Float).
AppendChildAsInt [▶ 162]	Appends a new node below an existing node (Integer).
AppendChildAsLint [▶ 162]	Appends a new node below an existing node (Integer64).
AppendChildAsUint [▶ 163]	Appends a new node below an existing node (Unsigned Integer).
AppendChildAsUlint [▶ 164]	Appends a new node below an existing node (Unsigned Integer64).
AppendCopy [▶ 165]	Appends a new node below an existing node.
AppendNode [▶ 165]	Appends a new node below an existing node.
Attributes [▶ 166]	Reads the attribute of a given XML node.
AttributeAsBool [▶ 167]	Returns the value of an attribute as data type Boolean.
AttributeAsDouble [▶ 167]	Returns the value of an attribute as data type Double.
AttributeAsFloat [▶ 168]	Returns the value of an attribute as data type Float.
AttributeAsInt [▶ 168]	Returns the value of an attribute as data type Integer.
AttributeAsLint [▶ 169]	Returns the value of an attribute as data type Integer64.
AttributeAsUint [▶ 169]	Returns the value of an attribute as data type Unsigned Integer.
AttributeAsUlint [▶ 170]	Returns the value of an attribute as data type Unsigned Integer64.
AttributeBegin [▶ 170]	Returns an iterator over all attributes of an XML node.
AttributeFromIterator [▶ 171]	Converts the current position of an iterator to an XML attribute object.
AttributeName [▶ 171]	Returns the name of a given attribute.
Attributes [▶ 172]	Used to navigate the DOM and returns an iterator for all attributes found at an XML node.
AttributeText [▶ 172]	Returns the text of a given attribute.
Begin [▶ 173]	Returns an iterator over all child elements of an XML node.
BeginByName [▶ 173]	Returns an iterator over all child elements of an XML node, starting at a particular element.
Child [▶ 174]	Used to navigate through the DOM. It returns a reference to the (first) child element of the current node.
ChildByAttribute [▶ 175]	Used to navigate through the DOM. It returns a reference to a child element in the XML document.
ChildByAttributeAndName [▶ 175]	Used to navigate through the DOM. It returns a reference to a child element in the XML document.
ChildByName [▶ 176]	Used to navigate through the DOM. It returns a reference to a child element in the XML document.

Name	Description
Children [▶ 177]	Used to navigate through the DOM. It returns an iterator for several child elements found in the XML document.
ChildrenByName [▶ 177]	Used to navigate through the DOM. It returns an iterator for several child elements found in the XML document.
ClearIterator [▶ 178]	This method resets iterators that have already been used so that they can be used again the next time the program is run.
Compare [▶ 179]	Checks two iterators for equality.
CopyAttributeText [▶ 179]	Reads the value of an XML attribute and writes it to a variable of data type String.
CopyDocument [▶ 180]	Copies the contents of the DOM memory into a variable of data type String.
CopyNodeText [▶ 181]	Reads the value of an XML node and writes it to a variable of data type String.
CopyNodeXml [▶ 181]	Reads the XML structure of an XML node and writes it to a variable of data type String.
End	
FirstNodeByPath [▶ 182]	Navigates through an XML document using a path that was transferred to the method.
GetAttributeTextLength [▶ 183]	Returns the length of the value of an XML attribute.
GetDocumentLength [▶ 183]	Returns the length of an XML document in bytes.
GetDocumentNode [▶ 184]	Returns the root node of an XML document.
GetNodeTextLength [▶ 184]	Returns the length of the value of an XML node.
GetNodeXmlLength [▶ 184]	Returns the length of the XML structure of an XML node.
GetRootNode [▶ 185]	Returns a reference to the first XML node in the XML document.
InsertAttributeCopy [▶ 185]	Inserts an attribute to an XML node, copying the name and value of an existing attribute.
InsertAttribute [▶ 186]	Inserts an attribute to an XML node.
InsertChild [▶ 187]	Inserts a node to an existing XML node.
InsertCopy [▶ 187]	Inserts a new node to an existing XML node and copies an existing node.
IsEnd [▶ 188]	Checks whether a given XML iterator is at the end of the iteration that is to be performed.
LoadDocumentFromFile [▶ 188]	Loads a XML document from a file.
NewDocument [▶ 189]	Generates a new empty XML document in the DOM memory.
Next [▶ 190]	Sets an XML iterator for the next object that is to be processed.
NextAttribute [▶ 190]	Returns the next attribute for a given XML attribute.
NextByName [▶ 191]	Sets an XML iterator for the next object that is to be processed, which is identified by its name.
NextSibling [▶ 191]	Returns the next direct node for a given XML node at the same XML level.
NextSiblingByName [▶ 192]	Returns the next direct node for a given XML node with a particular name at the same XML level.
Node [▶ 193]	Used in conjunction with an iterator to navigate through the DOM.
NodeAsBool [▶ 193]	Returns the text of an XML node as data type Boolean.
NodeAsDouble [▶ 194]	Returns the text of an XML node as data type Double.
NodeAsFloat [▶ 194]	Returns the text of an XML node as data type Float.
NodeAsInt [▶ 195]	Returns the text of an XML node as data type Integer.
NodeAsLint [▶ 195]	Returns the text of an XML node as data type Integer64.
NodeAsUint [▶ 196]	Returns the text of an XML node as data type Unsigned Integer.
NodeAsUlint [▶ 196]	Returns the text of an XML node as data type Unsigned Integer64.

Name	Description
NodeName [▶ 197]	Returns the name of an XML node.
NodeText [▶ 197]	Returns the text of an XML node.
ParseDocument [▶ 198]	Loads an XML object into the DOM memory for further processing.
RemoveAttribute	
RemoveAttributeByName	
RemoveChild [▶ 198]	Removes an XML child node from a given XML node.
RemoveChildByName [▶ 199]	Removes an XML child node from a given XML node.
SaveDocumentToFile [▶ 199]	Saves the current XML document in a file.
SetAttribute [▶ 200]	Sets the value of an attribute.
SetAttributeAsBool [▶ 201]	Sets the value of an attribute (Boolean).
SetAttributeAsDouble [▶ 202]	Sets the value of an attribute (double).
SetAttributeAsFloat [▶ 202]	Sets the value of an attribute (float).
SetAttributeAsInt [▶ 203]	Sets the value of an attribute (integer).
SetAttributeAsLint [▶ 203]	Sets the value of an attribute (Integer64).
SetAttributeAsUint [▶ 204]	Sets the value of an attribute (Unsigned Integer).
SetAttributeAsUlint [▶ 204]	Sets the value of an attribute (Unsigned Integer64).
SetChild [▶ 205]	Sets the value of an XML node (String).
SetChildAsBool [▶ 205]	Sets the value of an XML node (Boolean).
SetChildAsDouble [▶ 206]	Sets the value of an XML node (Double).
SetChildAsFloat [▶ 206]	Sets the value of an XML node (Float).
SetChildAsInt [▶ 207]	Sets the value of an XML node (Integer).
SetChildAsLint [▶ 208]	Sets the value of an XML node (Integer64).
SetChildAsUint [▶ 208]	Sets the value of an XML node (Unsigned Integer).
SetChildAsUlint [▶ 209]	Sets the value of an XML node (Unsigned Integer64).

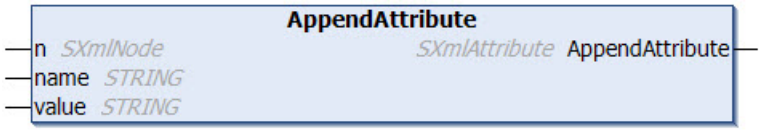
Requirements

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

Also see about this

- AppendAttribute [▶ 152]
- AppendAttributeAsBool [▶ 153]
- AppendAttributeAsDouble [▶ 154]
- AppendAttributeAsFloat [▶ 155]
- AppendAttributeAsInt [▶ 155]
- AppendAttributeAsLint [▶ 156]
- AppendAttributeAsUint [▶ 157]
- AppendAttributeAsUlint [▶ 157]
- AppendAttributeCopy [▶ 158]
- AppendChild [▶ 159]
- AppendChildAsBool [▶ 160]
- AppendChildAsDouble [▶ 160]
- AppendChildAsFloat [▶ 161]
- AppendChildAsInt [▶ 162]

4.7.1 AppendAttribute



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

Syntax

```
METHOD AppendAttribute : SXmlAttribute
VAR_INPUT
  n     : SXmlNode;
```

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

Return value

Name	Type
AppendAttribute	SXmlAttribute

Inputs

Name	Type
n	SXmlNode

/ Inputs/Outputs

Name	Type
name	STRING
value	STRING

Sample call:

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

4.7.2 AppendAttributeAsBool

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

Syntax

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

Return value

Name	Type
AppendAttributeAsBool	SXmlAttribute

Inputs

Name	Type
n	SXmlNode
value	BOOL

/ Inputs/Outputs

Name	Type
name	STRING

Sample call:

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

4.7.3 AppendAttributeAsDouble

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

Syntax

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

Return value

Name	Type
AppendAttributeAsDouble	SXmlAttribute

Inputs

Name	Type
n	SXmlNode
value	LREAL

/ Inputs/Outputs

Name	Type
name	STRING

Sample call:

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

4.7.4 AppendAttributeAsFloat

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

Syntax

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

Return value

Name	Type
AppendAttributeAsFloat	SXmlAttribute

Inputs

Name	Type
n	SXmlNode
value	REAL

/ Inputs/Outputs

Name	Type
name	STRING

Sample call:

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

4.7.5 AppendAttributeAsInt

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

Syntax

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

```
END_VAR
VAR_INPUT
  value : DINT;
END_VAR
```

Return value

Name	Type
AppendAttributeAsInt	SXmlAttribute

Inputs

Name	Type
n	SXmlNode
value	DINT

/ Inputs/Outputs

Name	Type
name	STRING

Sample call:

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

4.7.6 AppendAttributeAsLint

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

Syntax

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

Return value

Name	Type
AppendAttributeAsLint	SXmlAttribute

Inputs

Name	Type
n	SXmlNode
value	LINT

/ Inputs/Outputs

Name	Type
name	STRING

Sample call:

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

4.7.7 AppendAttributeAsUint

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

Syntax

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

Return value

Name	Type
AppendAttributeAsUint	SXmlAttribute

Inputs

Name	Type
n	SXmlNode
value	UDINT

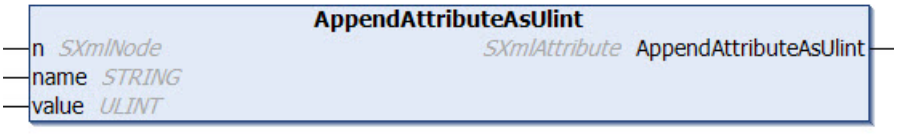
/ Inputs/Outputs

Name	Type
name	STRING

Sample call:

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

4.7.8 AppendAttributeAsUlint



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

Syntax

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

Return value

Name	Type
AppendAttributeAsUlint	SXmlAttribute

Inputs

Name	Type
n	SXmlNode
value	ULINT

/ Inputs/Outputs

Name	Type
name	STRING

Sample call:

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

4.7.9 AppendAttributeCopy

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

Syntax

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

Return value

Name	Type
AppendAttributeCopy	SXmlAttribute

Inputs

Name	Type
n	SXmlNode
copy	SXmlAttribute

Sample call:

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

4.7.10 AppendChild

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

Syntax

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

Return value

Name	Type
AppendChild	SXmlNode

Inputs

Name	Type
n	SXmlNode
cdata	BOOL

/ Inputs/Outputs

Name	Type
name	STRING
value	STRING

Sample call:

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

4.7.11 AppendChildAsBool

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

Syntax

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

Return value

Name	Type
AppendChildAsBool	SXmlNode

Inputs

Name	Type
n	SXmlNode
value	BOOL

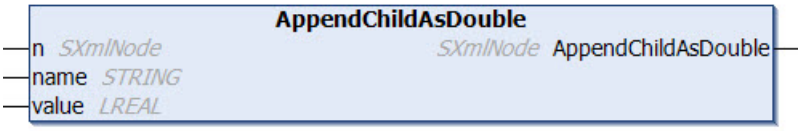
/ Inputs/Outputs

Name	Type
name	STRING

Sample call:

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

4.7.12 AppendChildAsDouble



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

Syntax

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

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

Return value

Name	Type
AppendChildAsDouble	SXmlNode

Inputs

Name	Type
n	SXmlNode
value	LREAL

/ Inputs/Outputs

Name	Type
name	STRING

Sample call:

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

4.7.13 AppendChildAsFloat

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

Syntax

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

Return value

Name	Type
AppendChildAsFloat	SXmlNode

Inputs

Name	Type
n	SXmlNode
value	REAL

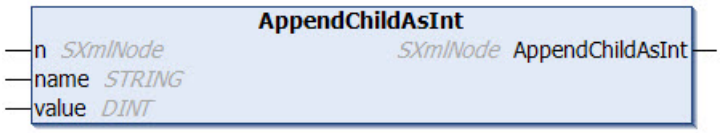
Inputs/Outputs

Name	Type
name	STRING

Sample call:

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

4.7.14 AppendChildAsInt



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

Syntax

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

Return value

Name	Type
AppendChildAsInt	SXmlNode

Inputs

Name	Type
n	SXmlNode
value	DINT

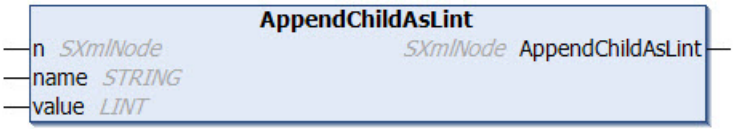
Inputs/Outputs

Name	Type
name	STRING

Sample call:

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

4.7.15 AppendChildAsLint



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

Syntax

```
METHOD AppendChildAsLint : SXmlNode
VAR_INPUT
  n     : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name  : STRING;
END_VAR
VAR_INPUT
  value : LINT;
END_VAR
```

Return value

Name	Type
AppendChildAsLint	SXmlNode

Inputs

Name	Type
n	SXmlNode
value	LINT

Inputs/Outputs

Name	Type
name	STRING

Sample call:

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

4.7.16 AppendChildAsUint

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

Syntax

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

Return value

Name	Type
AppendChildAsUint	SXmlNode

Inputs

Name	Type
n	SXmlNode
value	UDINT

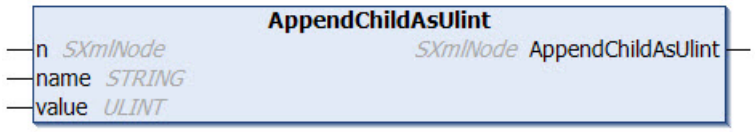
/ Inputs/Outputs

Name	Type
name	STRING

Sample call:

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

4.7.17 AppendChildAsUlint



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

Syntax

```
METHOD AppendChildAsUlint : SXmlNode
VAR_INPUT
  n     : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  name  : STRING;
END_VAR
VAR_INPUT
  value : ULINT;
END_VAR
```

Return value

Name	Type
AppendChildAsUlint	SXmlNode

Inputs

Name	Type
n	SXmlNode
value	ULINT

/ Inputs/Outputs

Name	Type
name	STRING

Sample call:

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

4.7.18 AppendCopy

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

Syntax

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

Return value

Name	Type
AppendCopy	SXmlNode

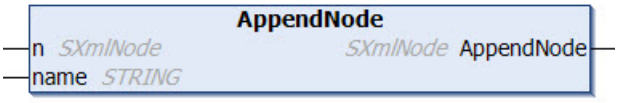
Inputs

Name	Type
n	SXmlNode
copy	SXmlNode

Sample call:

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

4.7.19 AppendNode



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

Syntax

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

Return value

Name	Type
AppendNode	SXmlNode

Inputs

Name	Type
n	SXmlNode

/ Inputs/Outputs

Name	Type
name	STRING

Sample call:

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

4.7.20 Attributes

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

Syntax

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

Return value

Name	Type
Attributes	SXmlAttribute

Inputs

Name	Type
n	SXmlNode

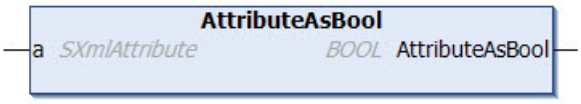
/ Inputs/Outputs

Name	Type
name	STRING

Sample call:

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

4.7.21 AttributeAsBool



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

Syntax

```
METHOD AttributeAsBool : BOOL
VAR_INPUT
  a : SXmlAttribute;
END_VAR
```

Return value

Name	Type
AttributeAsBool	BOOL

Inputs

Name	Type
a	SXmlAttribute

Sample call:

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

4.7.22 AttributeAsDouble

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

Syntax

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

Return value

Name	Type
AttributeAsDouble	LREAL

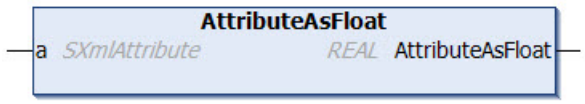
Inputs

Name	Type
a	SXmlAttribute

Sample call:

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

4.7.23 AttributeAsFloat



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

Syntax

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

Return value

Name	Type
AttributeAsFloat	REAL

Inputs

Name	Type
a	SXmlAttribute

Sample call:

```
rValue := fbXml.AttributeAsFloat(xmlAttr);
```

4.7.24 AttributeAsInt

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

Syntax

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

Return value

Name	Type
AttributeAsInt	DINT

Inputs

Name	Type
a	SXmlAttribute

Sample call:

```
nValue := fbXml.AttributeAsInt(xmlAttr);
```

4.7.25 AttributeAsLint

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

Syntax

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

Return value

Name	Type
AttributeAsLint	LINT

Inputs

Name	Type
a	SXmlAttribute

Sample call:

```
nValue := fbXml.AttributeAsLint(xmlAttr);
```

4.7.26 AttributeAsUint

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

Syntax

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

Return value

Name	Type
AttributeAsUint	UDINT

Inputs

Name	Type
a	SXmlAttribute

Sample call:

```
nValue := fbXml.AttributeAsUint(xmlAttr);
```

4.7.27 AttributeAsUlint

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

Syntax

```
METHOD AttributeAsUlint : ULINT
VAR_INPUT
  a : SXmlAttribute;
END_VAR
```

Return value

Name	Type
AttributeAsUlint	ULINT

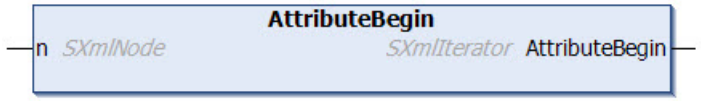
Inputs

Name	Type
a	SXmlAttribute

Sample call:

```
nValue := fbXml.AttributeAsUlint(xmlAttr);
```

4.7.28 AttributeBegin



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

Syntax

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

Return value

Name	Type
AttributeBegin	SXmlIterator

Inputs

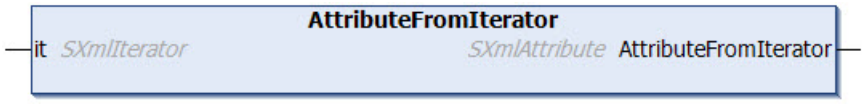
Name	Type
n	SXmlNode

Sample call:

```
xmlIterator := fbXml.AttributeBegin(xmlNode);
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlAttr     := fbXml.AttributeFromIterator(xmlIterator);
```

```
  nAttrValue  := fbXml.AttributeAsInt(xmlAttr);
  xmlIterator := fbXml.Next(xmlIterator);
END_WHILE
```

4.7.29 AttributeFromIterator



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

Syntax

```
METHOD AttributeFromIterator : SXmlAttribute
VAR_INPUT
  it : SXmlIterator;
END_VAR
```

Return value

Name	Type
AttributeFromIterator	SXmlAttribute

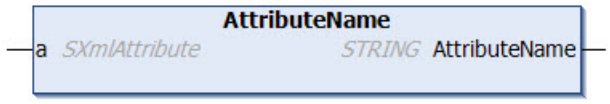
Inputs

Name	Type
it	SXmlIterator

Sample call:

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

4.7.30 AttributeName



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

Syntax

```
METHOD AttributeName : STRING
VAR_INPUT
  a : SXmlAttribute;
END_VAR
```

Return value

Name	Type
AttributeName	STRING

Inputs

Name	Type
a	SXmlAttribute

Sample call:

```
sName := fbXml.AttributeName(xmlAttr);
```

4.7.31 Attributes

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

Syntax

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

Return value

Name	Type
Attributes	SXmlAttribute

Inputs

Name	Type
it	REFERENCE TO SXmlIterator

Sample call:

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

4.7.32 AttributeText

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

Syntax

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

Return value

Name	Type
AttributeText	STRING(255)

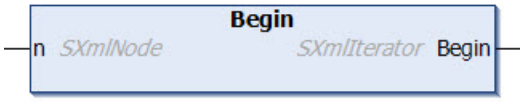
Inputs

Name	Type
a	SXmlAttribute

Sample call:

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

4.7.33 Begin



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

Syntax

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

Return value

Name	Type
Begin	SXmlIterator

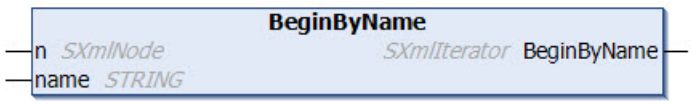
Inputs

Name	Type
n	SXmlNode

Sample call:

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

4.7.34 BeginByName



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

Syntax

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

Return value

Name	Type
BeginByName	SXmlIterator

Inputs

Name	Type
n	SXmlNode

/ Inputs/Outputs

Name	Type
name	STRING

Sample call:

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

4.7.35 Child

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

Syntax

```
METHOD ChildByName : SXmlNode
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Return value

Name	Type

Inputs

Name	Type
n	SXmlNode

Sample call:

```
xmlChild := fbXml.Child(xmlNode);
```

4.7.36 ChildByAttribute

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

Syntax

```
METHOD ChildByAttribute : SXmlNode
VAR_INPUT
  n     : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  attr  : STRING;
  value : STRING;
END_VAR
```

Return value

Name	Type
ChildByAttribute	SXmlNode

Inputs

Name	Type
n	SXmlNode

/ Inputs/Outputs

Name	Type
attr	STRING
value	STRING

Sample call:

```
xmlMachine1 := fbXml.ChildByAttribute(xmlMachines, 'Type', '1');
```

4.7.37 ChildByAttributeAndName

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

Syntax

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

Return value

Name	Type
ChildByAttributeAndName	SXmlNode

Inputs

Name	Type
n	SXmlNode

/ Inputs/Outputs

Name	Type
attr	STRING
value	STRING
child	STRING

Sample call:

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

4.7.38 ChildByName

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

Syntax

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

Return value

Name	Type
ChildByName	SXmlNode

Inputs

Name	Type
n	SXmlNode

/ Inputs/Outputs

Name	Type
name	STRING

Sample call:

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

4.7.39 Children

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

Syntax

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

Return value

Name	Type
Children	SXmlNode

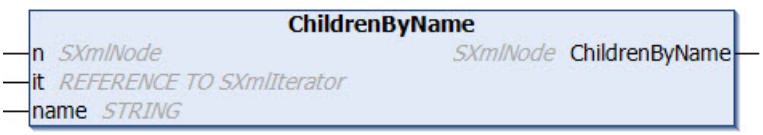
Inputs

Name	Type
n	SXmlNode

Sample call:

```
xmlRet := fbXml.Children(xmlNode, xmlIterator);
WHILE NOT fbXml.IsEnd(xmlIterator) DO
  xmlMachineNodeRef  := fbXml.Node(xmlIterator);
  xmlMachineNodeText := fbXml.NodeText(xmlMachineNodeRef);
  xmlIterator        := fbXml.Next(xmlIterator);
END_WHILE
```

4.7.40 ChildrenByName



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

Syntax

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

Return value

Name	Type
ChildrenByName	SXmlNode

Inputs

Name	Type
n	SXmlNode
it	REFERENCE TO SXmlIterator

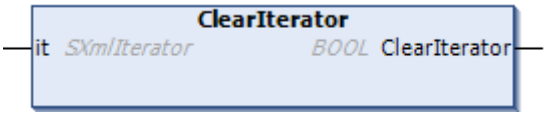
/ Inputs/Outputs

Name	Type
name	STRING

Sample call:

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

4.7.41 ClearIterator



This method resets iterators that have already been used so that they can be used again the next time the program is run.

Syntax

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

Return value

Name	Type
ClearIterator	BOOL

Inputs

Name	Type
it	SXmlIterator

Sample call:

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

Requirements

TwinCAT version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4024.56	x86, x64, ARM	Tc3_JsonXml 3.4.5.0

4.7.42 Compare

This method checks two iterators for equality.

Syntax

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

Return value

Name	Type
Compare	BOOL

Inputs

Name	Type
It1	SXmlIterator
It2	SXmlIterator

Sample call:

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

4.7.43 CopyAttributeText

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

Syntax

```
METHOD CopyAttributeText : UDINT
VAR_INPUT
  a    : SXmlAttribute;
END_VAR
VAR_IN_OUT CONSTANT
```

```
  sXml : STRING;
END_VAR
VAR_INPUT
  nXml : UDINT;
END_VAR
```

Return value

Name	Type
CopyAttributeText	UDINT

Inputs

Name	Type
a	SXmlAttribute
nXml	UDINT

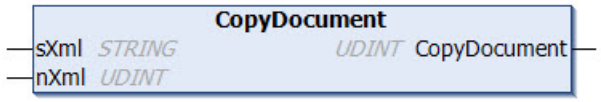
/ Inputs/Outputs

Name	Type
sXml	STRING

Sample call:

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

4.7.44 CopyDocument



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

Syntax

```
METHOD CopyDocument : UDINT
VAR_IN_OUT CONSTANT
  sXml : STRING;
END_VAR
VAR_INPUT
  nXml : UDINT;
END_VAR
```

Return value

Name	Type
CopyDocument	UDINT

Inputs

Name	Type
nXml	UDINT

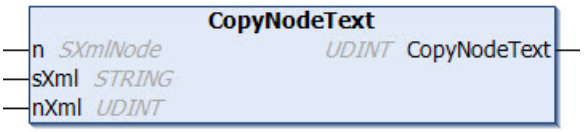
/ Inputs/Outputs

Name	Type
sXml	STRING

Sample call:

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

4.7.45 CopyNodeText



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

Syntax

```
METHOD CopyNodeText : UDINT
VAR_INPUT
  n    : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  sXml : STRING;
END_VAR
VAR_INPUT
  nXml : UDINT;
END_VAR
```

Return value

Name	Type
CopyNodeText	UDINT

Inputs

Name	Type
n	SXmlNode
nXml	UDINT

/ Inputs/Outputs

Name	Type
sXml	STRING

Sample call:

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

4.7.46 CopyNodeXml

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

Syntax

```
METHOD CopyNodeXml : UDINT
VAR_INPUT
  a    : SXmlNode;
END_VAR
VAR_IN_OUT CONSTANT
  sXml : STRING;
END_VAR
VAR_INPUT
  nXml : UDINT;
END_VAR
```

Return value

Name	Type
CopyNodeXml	UDINT

Inputs

Name	Type
a	SXmlNode
nXml	UDINT

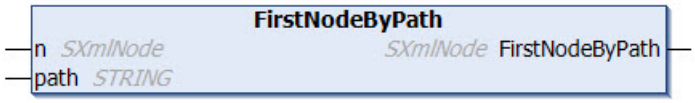
Inputs/Outputs

Name	Type
sXml	STRING

Sample call:

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

4.7.47 FirstNodeByPath



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

Syntax

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

Return value

Name	Type
FirstNodeByPath	SXmlNode

Inputs

Name	Type
n	SXmlNode
path	STRING

Sample call:

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

4.7.48 GetAttributeTextLength

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

Syntax

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

Return value

Name	Type
GetAttributeTextLength	UDINT

Inputs

Name	Type
a	SXmlAttribute

Sample call:

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

4.7.49 GetDocumentLength

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

Syntax

```
METHOD GetDocumentLength : UDINT
```

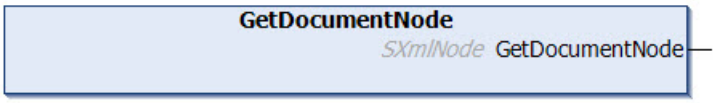
Return value

Name	Type
GetDocumentLength	UDINT

Sample call:

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

4.7.50 GetDocumentNode



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

Syntax

```
METHOD GetDocumentNode : SXmlNode
```

Return value

Name	Type
GetDocumentNode	SXmlNode

Sample call:

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

4.7.51 GetNodeTextLength

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

Syntax

```
METHOD GetNodeTextLength : UDINT
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Return value

Name	Type
GetNodeTextLength	UDINT

Inputs

Name	Type
n	SXmlNode

Sample call:

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

4.7.52 GetNodeXmlLength

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

Syntax

```
METHOD GetNodeXmlLength : UDINT
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Return value

Name	Type
GetNodeXmlLength	UDINT

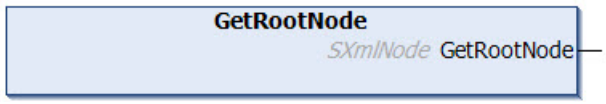
Inputs

Name	Type
n	SXmlNode

Sample call:

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

4.7.53 GetRootNode



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

Syntax

```
METHOD GetRootNode : SXmlNode
```

Return value

Name	Type
GetRootNode	SXmlNode

Sample call:

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

4.7.54 InsertAttributeCopy

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

Syntax

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

Return value

Name	Type
InsertAttributeCopy	SXmlAttribute

Inputs

Name	Type
n	SXmlNode
before	SXmlAttribute
copy	SXmlAttribute

Sample call:

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

4.7.55 InsertAttribute

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

Syntax

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

Return value

Name	Type
InsertAttribute	SXmlAttribute

Inputs

Name	Type
n	SXmlNode
before	SXmlAttribute

/ Inputs/Outputs

Name	Type
name	STRING

Sample call:

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

4.7.56 InsertChild

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

Syntax

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

Return value

Name	Type
InsertChild	SXmlNode

Inputs

Name	Type
n	SXmlNode
before	SXmlAttribute

/ Inputs/Outputs

Name	Type
name	STRING

Sample call:

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

4.7.57 InsertCopy

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

Syntax

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

Return value

Name	Type
InsertCopy	SXmlNode

Inputs

Name	Type
n	SXmlNode
before	SXmlNode
copy	SXmlNode

Sample call:

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

4.7.58 IsEnd

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

Syntax

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

Return value

Name	Type
IsEnd	BOOL

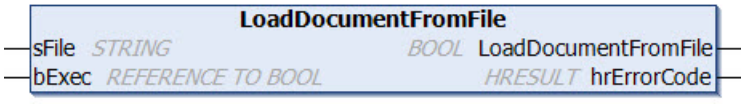
Inputs

Name	Type
nit	SXmlIterator

Sample call:

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

4.7.59 LoadDocumentFromFile



This method loads an XML document from a file. The absolute path to the file is transferred to the method as input parameter.

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

Syntax

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

Return value

Name	Type
LoadDocumentFromFile	BOOL

Inputs

Name	Type
bExec	REFERENCE TO BOOL

/ Inputs/Outputs

Name	Type
sFile	STRING

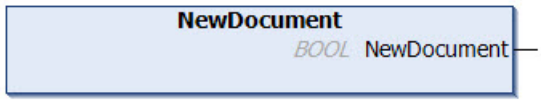
Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

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

4.7.60 NewDocument



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

Syntax

```
METHOD NewDocument : BOOL
```

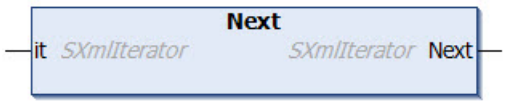
Return value

Name	Type
NewDocument	BOOL

Sample call:

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

4.7.61 Next



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

Syntax

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

Return value

Name	Type
Next	SXmlIterator

Inputs

Name	Type
it	SXmlIterator

Sample call:

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

4.7.62 NextAttribute

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

Syntax

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

Return value

Name	Type
NextAttribute	SXmlAttribute

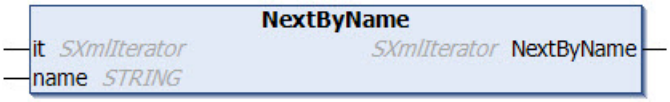
Inputs

Name	Type
a	SXmlAttribute

Sample call:

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

4.7.63 NextByName



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

Syntax

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

Return value

Name	Type
NextByName	SXmlIterator

Inputs

Name	Type
it	SXmlIterator

/ Inputs/Outputs

Name	Type
name	STRING

Sample call:

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

4.7.64 NextSibling

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

Syntax

```
METHOD NextSibling : SXmlNode
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Return value

Name	Type
NextSibling	SXmlNode

Inputs

Name	Type
n	SXmlNode

Sample call:

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

4.7.65 NextSiblingByName

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

Syntax

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

Return value

Name	Type
NextSiblingByName	SXmlNode

Inputs

Name	Type
n	SXmlNode

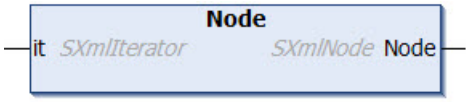
Inputs/Outputs

Name	Type
name	STRING

Sample call:

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

4.7.66 Node



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

Syntax

```
METHOD Node : SXmlNode
VAR_INPUT
  it : SXmlIterator;
END_VAR
```

Return value

Name	Type
Node	SXmlNode

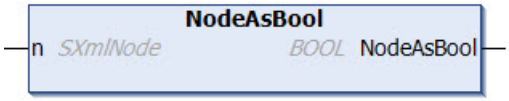
Inputs

Name	Type
it	SXmlIterator

Sample call:

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

4.7.67 NodeAsBool



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

Syntax

```
METHOD NodeAsBool : BOOL
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Return value

Name	Type
NodeAsBool	BOOL

Inputs

Name	Type
n	SXmlNode

Sample call:

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

4.7.68 NodeAsDouble

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

Syntax

```
METHOD NodeAsDouble : LREAL
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Return value

Name	Type
NodeAsDouble	LREAL

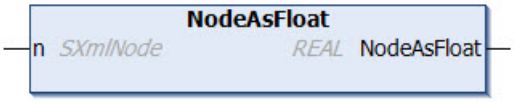
Inputs

Name	Type
n	SXmlNode

Sample call:

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

4.7.69 NodeAsFloat



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

Syntax

```
METHOD NodeAsFloat : REAL
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Return value

Name	Type
NodeAsFloat	REAL

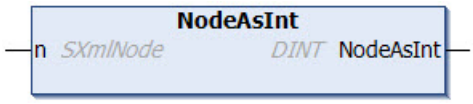
Inputs

Name	Type
n	SXmlNode

Sample call:

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

4.7.70 NodeAsInt



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

Syntax

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

Return value

Name	Type
NodeAsInt	DINT

Inputs

Name	Type
n	SXmlNode

Sample call:

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

4.7.71 NodeAsLint

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

Syntax

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

Return value

Name	Type
NodeAsLint	LINT

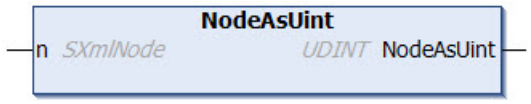
Inputs

Name	Type
n	SXmlNode

Sample call:

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

4.7.72 NodeAsUint



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

Syntax

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

Return value

Name	Type
NodeAsUint	UDINT

Inputs

Name	Type
n	SXmlNode

Sample call:

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

4.7.73 NodeAsUlint

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

Syntax

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

Return value

Name	Type
NodeAsUlint	ULINT

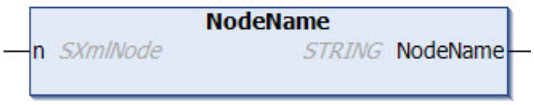
Inputs

Name	Type
n	SXmlNode

Sample call:

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

4.7.74 NodeName



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

Syntax

```
METHOD NodeName : STRING
VAR_INPUT
  n : SXmlNode;
END_VAR
```

Return value

Name	Type
NodeName	STRING

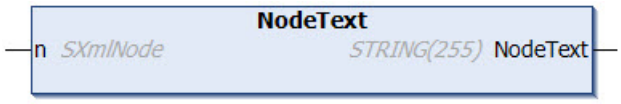
Inputs

Name	Type
n	SXmlNode

Sample call:

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

4.7.75 NodeText



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

If the text of an XML node is longer than 255 characters, the method CopyNodeText [▶ 181] must be used.

Syntax

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

Return value

Name	Type
NodeText	STRING(255)

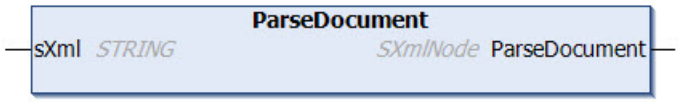
Inputs

Name	Type
n	SXmlNode

Sample call:

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

4.7.76 ParseDocument



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

Syntax

```
METHOD ParseDocument : SXmlNode
VAR_IN_OUT CONSTANT
  sXml : STRING;
END_VAR
```

Return value

Name	Type
ParseDocument	SXmlNode

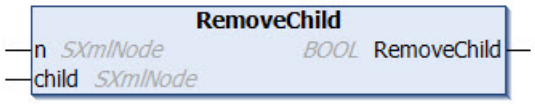
Inputs/Outputs

Name	Type
sXml	STRING

Sample call:

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

4.7.77 RemoveChild



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

Syntax

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

Return value

Name	Type
RemoveChild	BOOL

Inputs

Name	Type
n	SXmlNode
child	SXmlNode

Sample call:

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

4.7.78 RemoveChildByName

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

Syntax

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

Return value

Name	Type
RemoveChildByName	BOOL

Inputs

Name	Type
n	SXmlNode

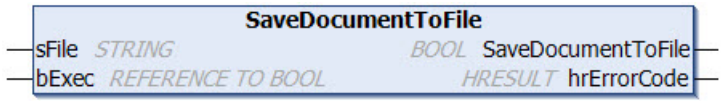
/ Inputs/Outputs

Name	Type
name	STRING

Sample call:

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

4.7.79 SaveDocumentToFile



This method saves the current XML document in a file. The absolute path to the file is transferred to the method as input parameter.

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

Syntax

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

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

Return value

Name	Type
SaveDocumentToFile	BOOL

Inputs

Name	Type
bExec	REFERENCE TO BOOL

/ Inputs/Outputs

Name	Type
sFile	STRING

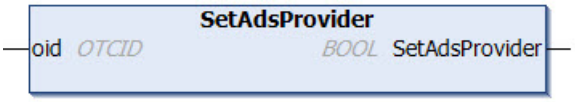
Outputs

Name	Type
hrErrorCode	HRESULT

Sample call:

```
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
```

Return value

Name	Type
SetAdsProvider	BOOL

/ Inputs/Outputs

Name	Type
oid	OTCID

4.7.81 SetAttribute

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

Syntax

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

Return value

Name	Type
SetAttribute	SXmlAttribute

Inputs

Name	Type
a	SXmlAttribute

Inputs/Outputs

Name	Type
value	STRING

Sample call:

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

4.7.82 SetAttributeAsBool

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

Syntax

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

Return value

Name	Type
SetAttributeAsBool	SXmlAttribute

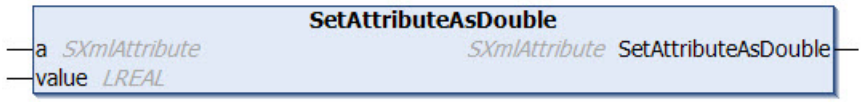
Inputs

Name	Type
a	SXmlAttribute
value	BOOL

Sample call:

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

4.7.83 SetAttributeAsDouble



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

Syntax

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

Return value

Name	Type
SetAttributeAsDouble	SXmlAttribute

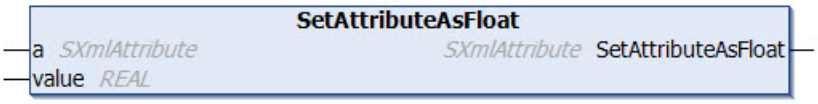
Inputs

Name	Type
a	SXmlAttribute
value	LREAL

Sample call:

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

4.7.84 SetAttributeAsFloat



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

Syntax

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

Return value

Name	Type
SetAttributeAsFloat	SXmlAttribute

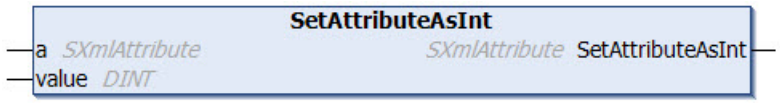
Inputs

Name	Type
a	SXmlAttribute
value	REAL

Sample call:

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

4.7.85 SetAttributeAsInt



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

Syntax

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

Return value

Name	Type
SetAttributeAsInt	SXmlAttribute

Inputs

Name	Type
a	SXmlAttribute
value	DINT

Sample call:

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

4.7.86 SetAttributeAsLint

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

Syntax

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

Return value

Name	Type
SetAttributeAsLint	SXmlAttribute

Inputs

Name	Type
a	SXmlAttribute
value	LINT

Sample call:

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

4.7.87 SetAttributeAsUint

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

Syntax

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

Return value

Name	Type
SetAttributeAsUint	SXmlAttribute

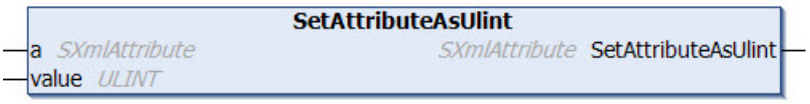
Inputs

Name	Type
a	SXmlAttribute
value	UDINT

Sample call:

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

4.7.88 SetAttributeAsUlint



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

Syntax

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

Return value

Name	Type
SetAttributeAsUlint	SXmlAttribute

Inputs

Name	Type
a	SXmlAttribute
value	ULINT

Sample call:

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

4.7.89 SetChild

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

Syntax

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

Return value

Name	Type
SetChild	SXmlNode

Inputs

Name	Type
n	SXmlNode
cdata	BOOL

/ Inputs/Outputs

Name	Type
value	STRING

Sample call:

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

4.7.90 SetChildAsBool

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

Syntax

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

Return value

Name	Type
SetChildAsBool	SXmlNode

Inputs

Name	Type
n	SXmlNode
value	BOOL

Sample call:

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

4.7.91 SetChildAsDouble

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

Syntax

```
METHOD SetChildAsDouble : SXmlNode
VAR_INPUT
  n     : SXmlNode;
  value : LREAL;
END_VAR
```

Return value

Name	Type
SetChildAsDouble	SXmlNode

Inputs

Name	Type
n	SXmlNode
value	LREAL

Sample call:

```
xmlNode := fbXml.SetChildAsDouble(xmlExistingNode, 42.42);
```

4.7.92 SetChildAsFloat

This method sets the value of an XML node. The value is transferred to the method as input parameter of data type Float.

Syntax

```
METHOD SetChildAsFloat : SXmlNode
VAR_INPUT
  n     : SXmlNode;
  value : REAL;
END_VAR
```

Return value

Name	Type
SetChildAsFloat	SXmlNode

Inputs

Name	Type
n	SXmlNode
value	REAL

Sample call:

```
xmlNode := fbXml.SetChildAsFloat(xmlExistingNode, 42.42);
```

4.7.93 SetChildAsInt

This method sets the value of an XML node. The value is transferred to the method as input parameter of data type Integer.

Syntax

```
METHOD SetChildAsInt : SXmlNode
VAR_INPUT
  n     : SXmlNode;
  value : DINT;
END_VAR
```

Return value

Name	Type
SetChildAsInt	SXmlNode

Inputs

Name	Type
n	SXmlNode
value	DINT

Sample call:

```
xmlNode := fbXml.SetChildAsInt(xmlExistingNode, 42);
```

4.7.94 SetChildAsLint

This method sets the value of an XML node. The value is transferred to the method as input parameter of data type Integer64.

Syntax

```
METHOD SetChildAsLint : SXmlNode
VAR_INPUT
  n     : SXmlNode;
  value : LINT;
END_VAR
```

Return value

Name	Type
SetChildAsLint	SXmlNode

Inputs

Name	Type
n	SXmlNode
value	LINT

Sample call:

```
xmlNode := fbXml.SetChildAsLint(xmlExistingNode, 42);
```

4.7.95 SetChildAsUint

This method sets the value of an XML node. The value is transferred to the method as input parameter of data type Unsigned Integer.

Syntax

```
METHOD SetChildAsUint : SXmlNode
VAR_INPUT
  n     : SXmlNode;
  value : UDINT;
END_VAR
```

Return value

Name	Type
SetChildAsUint	SXmlNode

Inputs

Name	Type
n	SXmlNode
value	UDINT

Sample call:

```
xmlNode := fbXml.SetChildAsUint(xmlExistingNode, 42);
```

4.7.96 SetChildAsUlint

This method sets the value of an XML node. The value is transferred to the method as input parameter of data type Unsigned Integer64.

Syntax

```
METHOD SetChildAsUlint : SXmlNode
VAR_INPUT
  n     : SXmlNode;
  value : ULINT;
END_VAR
```

Return value

Name	Type
SetChildAsUlint	SXmlNode

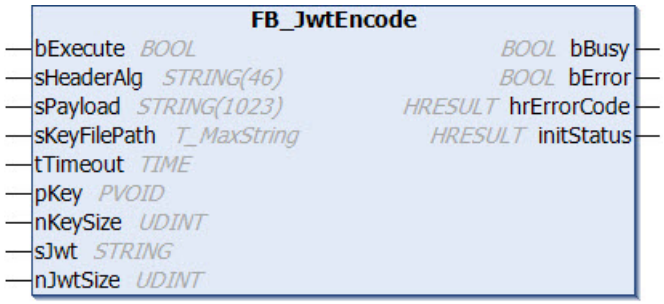
Inputs

Name	Type
n	SXmlNode
value	ULINT

Sample call:

```
xmlNode := fbXml.SetChildAsUlint(xmlExistingNode, 42);
```

4.8 FB_JwtEncode



The function block enables the creation and signing of a JSON Web Token (JWT).

Syntax

Definition:

```
FUNCTION_BLOCK FB_JwtEncode
VAR_INPUT
  bExecute       : BOOL;
  sHeaderAlg     : STRING(46);
  sPayload       : STRING(1023);
  sKeyFilePath   : STRING(511);
  tTimeout       : TIME;
  pKey           : PVOID;
  nKeySize       : UDINT;
```

```
  nJwtSize       : UDINT;
END_VAR
VAR_IN_OUT CONSTANT
  sJwt           : STRING;
END_VAR
VAR_OUTPUT
  bBusy          : BOOL;
  bError         : BOOL;
  hrErrorCode    : HRESULT;
  initStatus     : HRESULT;
END_VAR
```

Inputs

Name	Type	Description
bExecute	BOOL	A rising edge activates processing of the function block.
sHeaderAlg	STRING(46)	The algorithm to be used for the JWT header, e.g. RS256.
sPayload	STRING(1023)	The JWT payload to be used.
sKeyFilePath	STRING(511)	Path to the private key to be used for the signature of the JWT.
tTimeout	TIME	ADS timeout, which is used internally for file access to the private key.
pKey	PVOID	Buffer for the private key to be read.
nKeySize	UDINT	Maximum size of the buffer.
sJwt	STRING	Contains the fully coded and signed JWT after the function block has been processed.
nJwtSize	UDINT	Size of the generated JWT including zero termination.

Outputs

Name	Type	Description
bBusy	BOOL	Is TRUE as long as processing of the function block is in progress.
bError	BOOL	Becomes TRUE as soon as an error situation occurs.
hrErrorCode	HRESULT	Returns an error code if the `bError` output is set. An explanation of the possible error codes can be found in the Appendix.
initStatus	HRESULT	Returns an error code in case of a failed initialization of the function block.

Requirements

TwinCAT version	Hardware	Libraries to be integrated
TwinCAT 3.1, Build 4024.4	x86, x64, ARM	Tc3_JsonXml 3.3.6.0

5 Interfaces

5.1 ITcJsonSaxHandler

5.1.1 OnBool

This callback method is triggered if a value of the data type BOOL was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnBool : HRESULT
VAR_INPUT
    value : BOOL;
END_VAR
```

5.1.2 OnDint

This callback method is triggered if a value of the data type DINT was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnDint : HRESULT
VAR_INPUT
    value : DINT;
END_VAR
```

5.1.3 OnEndArray

This callback method is triggered if a square closing bracket, which corresponds to the JSON synonym for an ending array, was found at the position of the SAX reader. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnEndArray : HRESULT
```

5.1.4 OnEndObject

This callback method is triggered if a curly closing bracket, which corresponds to the JSON synonym for an ending object, was found at the position of the SAX reader. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnEndObject : HRESULT
```

5.1.5 OnKey

This callback method is triggered if a property was found at the position of the SAX reader. The property name lies on the input/output parameter key and its length on the input parameter len. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnKey : HRESULT
VAR_IN_OUT CONSTANT
    key : STRING;
END_VAR
```

```
VAR_INPUT
    len : UDINT;
END_VAR
```

5.1.6 OnLint

This callback method is triggered if a value of the data type LINT was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnLint : HRESULT
VAR_INPUT
    value : LINT;
END_VAR
```

5.1.7 OnLreal

This callback method is triggered if a value of the data type LREAL was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnLreal : HRESULT
VAR_INPUT
    value : LREAL;
END_VAR
```

5.1.8 OnNull

This callback method is triggered if a NULL value was found at the position of the SAX reader. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnNull : HRESULT
```

5.1.9 OnStartArray

This callback method is triggered if a square opening bracket, which corresponds to the JSON synonym for a starting array, was found at the position of the SAX reader. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnStartArray : HRESULT
```

5.1.10 OnStartObject

This callback method is triggered if a curly opening bracket, which corresponds to the JSON synonym for a starting object, was found at the position of the SAX reader. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnStartObject : HRESULT
```

5.1.11 OnString

This callback method is triggered if a value of the data type STRING was found at the position of the SAX reader. The In/Out parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnString : HRESULT
VAR_IN_OUT CONSTANT
    value : STRING;
END_VAR
VAR_INPUT
    len   : UDINT;
END_VAR
```

5.1.12 OnUdint

This callback method is triggered if a value of the data type UDINT was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnUdint : HRESULT
VAR_INPUT
    value : UDINT;
END_VAR
```

5.1.13 OnUlint

This callback method is triggered if a value of the data type ULINT was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnUlint : HRESULT
VAR_INPUT
    value : ULINT;
END_VAR
```

5.2 ITcJsonSaxValues

5.2.1 OnBoolValue

This callback method is triggered if a value of the data type BOOL was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnBoolValue : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
    value : BOOL;
END_VAR
```

5.2.2 OnDintValue

This callback method is triggered if a value of the data type DINT was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnDintValue : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
    value : DINT;
END_VAR
```

5.2.3 OnLintValue

This callback method is triggered if a value of the data type LINT was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnLintValue : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
    value : LINT;
END_VAR
```

5.2.4 OnLrealValue

This callback method is triggered if a value of the data type LREAL was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnLrealValue : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
    value : LREAL;
END_VAR
```

5.2.5 OnNullValue

This callback method is triggered if a NULL value was found at the position of the SAX reader. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnNull : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
END_VAR
```

5.2.6 OnStringValue

This callback method is triggered if a value of the data type STRING was found at the position of the SAX reader. The input/output parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnStringValue : HRESULT
VAR_IN_OUT CONSTANT
    value : STRING;
END_VAR
VAR_INPUT
    len : UDINT;
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
END_VAR
```

5.2.7 OnUdintValue

This callback method is triggered if a value of the data type UDINT was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

Syntax

```
METHOD OnUdintValue : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
    value : UDINT;
END_VAR
```

5.2.8 OnUlintValue

This callback method is triggered if a value of the data type ULINT was found at the position of the SAX reader. The input parameter value contains the value found. The SAX parsing procedure is aborted by setting the return value HRESULT to S_FALSE.

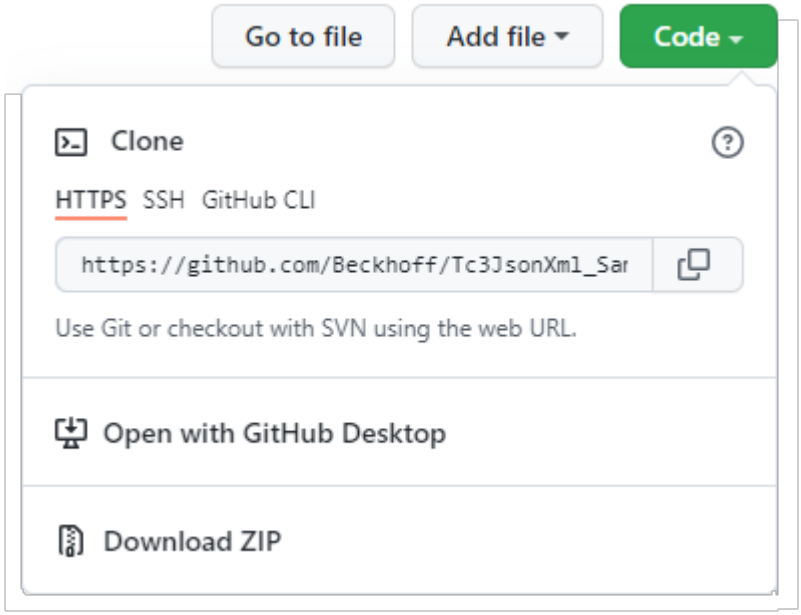
Syntax

```
METHOD OnUlintValue : HRESULT
VAR_INPUT
    level : UDINT;
    infos : POINTER TO TcJsonLevelInfo;
    value : ULINT;
END_VAR
```

6 Samples

Downloads

Sample code and configurations for this product can be obtained from the corresponding repository on GitHub: https://github.com/Beckhoff/Tc3JsonXml_Samples. There you have the option to clone the repository or download a ZIP file containing the sample.



6.1 Tc3JsonXmlSampleJsonDataType

Sample of the automatic conversion of structures into a JSON message

This sample illustrates how a data structure can be converted into a JSON message (and vice versa). In the conversion the layout of a structure is converted one-to-one into a corresponding JSON equivalent. Additional metadata can be created via PLC attributes on the member variables of the structure.

Layout of the data structure to be converted

```
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
```

Declaration range

```
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
```

Implementation range

Two ways of generating the JSON message are shown, starting with the instance fbJson of the function block FB_JsonSaxWriter. The GetDocument() method can be used with a JSON message with no more than 255 characters. However, the CopyDocument() method must be used with larger JSON messages.

```
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));
```

Resulting JSON message

```
{
  "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

As an alternative, the method AddJsonValueFromSymbol() can also be used to generate a JSON format directly from a data structure.

```
fbJson.ResetDocument();
fbJsonDataType.AddJsonValueFromSymbol(fbJson, 'ST_Values', SIZEOF(stValues), ADR(stValues));
sJsonDoc := fbJson.GetDocument();
fbJson.CopyDocument(sJsonDoc2, SIZEOF(sJsonDoc2));
```

The resulting JSON object looks like this:

```
{
  "Sensor1": 0.0,
  "Sensor2": 0,
  "Sensor3": 0
}
```

Conversion of a JSON message back to a data structure

The above samples show how a JSON object can be generated from a data structure in a simple manner. There is also a corresponding method in the Tc3_JsonXml library for the reverse process, i.e. the extraction of values from a (received) JSON object back into a data structure. This application is made possible by calling the method SetSymbolFromJson().

```
fbJsonDataType.SetSymbolFromJson(someJson, 'ST_Values', SIZEOF(stValuesReceive),
ADR(stValuesReceive));
```

The string variable sJsonDoc2 contains the JSON object, which is transferred into the structure instance stValuesReceive by calling the method.

Target data structure

The target data structure must match the structure of the JSON document. Otherwise SetSymbolFromJson() returns FALSE.

6.2 Tc3JsonXmlSampleJsonSaxReader

Sample of the parsing of JSON documents via SAX Reader

This sample illustrates how a JSON message can be run through programmatically. The function block FB_JsonSaxReader is used as the basis.

Declaration range

```
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
```

Implementation range

Through the calling of the Parse() method, the transfer of the JSON message as a STRING and the interface pointer to a function block instance that implements the interface ItcJsonSaxHandler, the SAX Reader is activated and the corresponding callback methods are run through.

```
fbJson.Parse(sJson := sJsonDoc, ipHdl := pJsonParse);
```

Callback methods

The callback methods are called on the instance of the function block that implements the interface ItcJsonSaxHandler. Each callback method represents a "found" element in the JSON message. For example, the callback method OnStartObject() is called as soon as an opening curly bracket has been detected. According to the example JSON message mentioned above, therefore, the following callback methods are run through in this order:

1. OnStartObject(), due to the first opening curly bracket
2. OnKey(), due to the property "Values"
3. OnStartObject(), due to the second opening curly bracket
4. OnKey(), due to the property "Timestamp"
5. OnString(), due to the value of the property "Timestamp"
6. OnKey(), due to the property "Sensor1"
7. OnLreal(), due to the value of the property "Sensor1"
8. OnKey(), due to the property "Sensor2"
9. OnUdint(), due to the value of the property "Sensor2"
10. OnEndObject(), due to the first closing curly bracket
11. OnEndObject(), due to the second closing curly bracket

Within the callback methods the current state is defined and saved via an instance of the enum E_JsonStates. This can also be used to determine whether the JSON message is valid. For example, if the callback method OnLreal() is called and the state is not the expected State 70 (JSON_STATE_ONLREAL), the return value S_FALSE can be returned to the method. The SAX Reader then automatically cancels the further processing.

6.3 Tc3JsonXmlSampleJsonSaxWriter

Sample of the creation of JSON documents via SAX Writer

This sample illustrates how a JSON message can be created over the DAX mechanism. The function block FB_JsonSaxWriter is used as the basis.

Declaration range

```
PROGRAM MAIN
VAR
  dtTimestamp : DATE_AND_TIME := DT#2017-04-04-12:42:42;
```

```
  fbJson      : FB_JsonSaxWriter;
  sJsonDoc    : STRING(255);
END_VAR
```

Implementation range

The SAX mechanism runs sequentially through the JSON document to be created, i.e. the corresponding elements are run though and created one after the other.

```
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();
```

Resulting JSON message

```
{
  "Timestamp": "2017-04-04T12:42:42",
  "Values": {
    "Sensor1": 42.42,
    "Sensor2": 42,
    "Sensor3": true
  }
}
```

6.4 Tc3JsonXmlSampleJsonDomReader

This sample illustrates how a JSON message can be run through programmatically on the basis of DOM. The function block FB_JsonDomParser is used as the basis.

Declaration range

```
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
```

Implementation range

The JSON message is loaded into the DOM tree using the ParseDocument() method. You can subsequently check whether it contains a certain property using the HasMember() method. The FindMember() method selects the property. The GetString() method extracts its value.

```
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
```

The use of the method HasMember() is not absolutely necessary, since the method FindMember() already returns 0 if a property was not found. The code shown above can also be implemented as follows:

```
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
```

Nested JSON objects

The approach is similar with nested JSON objects. Since the entire document is located in the DOM, it is simple to navigate. Let's take a JSON object that looks like this:

```
sMessage : STRING(255) := '{"Values":{"serial":"G030PT028191AC4R"}}';
```

The property we are looking for is located in the sub-object "Values". The following code shows how to extract the property.

```
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

This sample illustrates how an XML document can be processed programmatically based on DOM. The function block FB_XmlDomParser is used as a basis.

Declaration range

```
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
```

Implementation range

The implementation section shows various options for parsing an XML document.

```
(* 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

This sample illustrates how an XML document can be created programmatically based on DOM. The function block FB_XmlDomParser is used as a basis.

Declaration range

```
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
```

Implementation range

The implementation section shows various options for creating an XML document.

```
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 Error Codes

7.1 ADS Return Codes

Grouping of error codes:
Global error codes: ADS Return Codes [▶ 223]... (0x9811_0000 ...)
Router error codes: ADS Return Codes [▶ 223]... (0x9811_0500 ...)
General ADS errors: ADS Return Codes [▶ 224]... (0x9811_0700 ...)
RTime error codes: ADS Return Codes [▶ 226]... (0x9811_1000 ...)

Global error codes

Hex	Dec	HRESULT	Name	Description
0x0	0	0x98110000	ERR_NOERROR	No error.
0x1	1	0x98110001	ERR_INTERNAL	Internal error.
0x2	2	0x98110002	ERR_NORTIME	No real time.
0x3	3	0x98110003	ERR_ALLOCLOCKEDMEM	Allocation locked – memory error.
0x4	4	0x98110004	ERR_INSERTMAILBOX	Mailbox full – the ADS message could not be sent. Reducing the number of ADS messages per cycle will help.
0x5	5	0x98110005	ERR_WRONGRECEIVEHMSG	Wrong HMSG.
0x6	6	0x98110006	ERR_TARGETPORTNOTFOUND	Target port not found – ADS server is not started or is not reachable.
0x7	7	0x98110007	ERR_TARGETMACHINENOTFOUND	Target computer not found – AMS route was not found.
0x8	8	0x98110008	ERR_UNKNOWNCMDID	Unknown command ID.
0x9	9	0x98110009	ERR_BADTASKID	Invalid task ID.
0xA	10	0x9811000A	ERR_NOIO	No IO.
0xB	11	0x9811000B	ERR_UNKNOWNAMSCMD	Unknown AMS command.
0xC	12	0x9811000C	ERR_WIN32ERROR	Win32 error.
0xD	13	0x9811000D	ERR_PORTNOTCONNECTED	Port not connected.
0xE	14	0x9811000E	ERR_INVALIDAMSLENGTH	Invalid AMS length.
0xF	15	0x9811000F	ERR_INVALIDAMSNETID	Invalid AMS Net ID.
0x10	16	0x98110010	ERR_LOWINSTLEVEL	Installation level is too low –TwinCAT 2 license error.
0x11	17	0x98110011	ERR_NODEBUGINTAVAILABLE	No debugging available.
0x12	18	0x98110012	ERR_PORTDISABLED	Port disabled – TwinCAT system service not started.
0x13	19	0x98110013	ERR_PORTALREADYCONNECTED	Port already connected.
0x14	20	0x98110014	ERR_AMSSYNC_W32ERROR	AMS Sync Win32 error.
0x15	21	0x98110015	ERR_AMSSYNC_TIMEOUT	AMS Sync Timeout.
0x16	22	0x98110016	ERR_AMSSYNC_AMSERROR	AMS Sync error.
0x17	23	0x98110017	ERR_AMSSYNC_NOINDEXINMAP	No index map for AMS Sync available.
0x18	24	0x98110018	ERR_INVALIDAMSPORT	Invalid AMS port.
0x19	25	0x98110019	ERR_NOMEMORY	No memory.
0x1A	26	0x9811001A	ERR_TCPSEND	TCP send error.
0x1B	27	0x9811001B	ERR_HOSTUNREACHABLE	Host unreachable.
0x1C	28	0x9811001C	ERR_INVALIDAMSFRAGMENT	Invalid AMS fragment.
0x1D	29	0x9811001D	ERR_TLSSEND	TLS send error – secure ADS connection failed.
0x1E	30	0x9811001E	ERR_ACCESSDENIED	Access denied – secure ADS access denied.

Router error codes

Hex	Dec	HRESULT	Name	Description
0x500	1280	0x98110500	ROUTERERR_NOLOCKEDMEMORY	Locked memory cannot be allocated.
0x501	1281	0x98110501	ROUTERERR_RESIZEMEMORY	The router memory size could not be changed.
0x502	1282	0x98110502	ROUTERERR_MAILBOXFULL	The mailbox has reached the maximum number of possible messages.
0x503	1283	0x98110503	ROUTERERR_DEBUGBOXFULL	The Debug mailbox has reached the maximum number of possible messages.
0x504	1284	0x98110504	ROUTERERR_UNKNOWNPORTTYPE	The port type is unknown.
0x505	1285	0x98110505	ROUTERERR_NOTINITIALIZED	The router is not initialized.
0x506	1286	0x98110506	ROUTERERR_PORTALREADYINUSE	The port number is already assigned.
0x507	1287	0x98110507	ROUTERERR_NOTREGISTERED	The port is not registered.
0x508	1288	0x98110508	ROUTERERR_NOMOREQUEUES	The maximum number of ports has been reached.
0x509	1289	0x98110509	ROUTERERR_INVALIDPORT	The port is invalid.
0x50A	1290	0x9811050A	ROUTERERR_NOTACTIVATED	The router is not active.
0x50B	1291	0x9811050B	ROUTERERR_FRAGMENTBOXFULL	The mailbox has reached the maximum number for fragmented messages.
0x50C	1292	0x9811050C	ROUTERERR_FRAGMENTTIMEOUT	A fragment timeout has occurred.
0x50D	1293	0x9811050D	ROUTERERR_TOBEREMOVED	The port is removed.

General ADS error codes

Hex	Dec	HRESULT	Name	Description
0x700	1792	0x98110700	ADSERR_DEVICE_ERROR	General device error.
0x701	1793	0x98110701	ADSERR_DEVICE_SRVNOTSUPP	Service is not supported by the server.
0x702	1794	0x98110702	ADSERR_DEVICE_INVALIDGRP	Invalid index group.
0x703	1795	0x98110703	ADSERR_DEVICE_INVALIDOFFSET	Invalid index offset.
0x704	1796	0x98110704	ADSERR_DEVICE_INVALIDACCESS	Reading or writing not permitted.
0x705	1797	0x98110705	ADSERR_DEVICE_INVALIDSIZE	Parameter size not correct.
0x706	1798	0x98110706	ADSERR_DEVICE_INVALIDDATA	Invalid data values.
0x707	1799	0x98110707	ADSERR_DEVICE_NOTREADY	Device is not ready to operate.
0x708	1800	0x98110708	ADSERR_DEVICE_BUSY	Device is busy.
0x709	1801	0x98110709	ADSERR_DEVICE_INVALIDCONTEXT	Invalid operating system context. This can result from use of ADS blocks in different tasks. It may be possible to resolve this through multitasking synchronization in the PLC.
0x70A	1802	0x9811070A	ADSERR_DEVICE_NOMEMORY	Insufficient memory.
0x70B	1803	0x9811070B	ADSERR_DEVICE_INVALIDPARM	Invalid parameter values.
0x70C	1804	0x9811070C	ADSERR_DEVICE_NOTFOUND	Not found (files, ...).
0x70D	1805	0x9811070D	ADSERR_DEVICE_SYNTAX	Syntax error in file or command.
0x70E	1806	0x9811070E	ADSERR_DEVICE_INCOMPATIBLE	Objects do not match.
0x70F	1807	0x9811070F	ADSERR_DEVICE_EXISTS	Object already exists.
0x710	1808	0x98110710	ADSERR_DEVICE_SYMBOLNOTFOUND	Symbol not found.
0x711	1809	0x98110711	ADSERR_DEVICE_SYMBOLVERSIONINVALID	Invalid symbol version. This can occur due to an online change. Create a new handle.
0x712	1810	0x98110712	ADSERR_DEVICE_INVALIDSTATE	Device (server) is in invalid state.
0x713	1811	0x98110713	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransMode not supported.
0x714	1812	0x98110714	ADSERR_DEVICE_NOTIFYHNDINVALID	Notification handle is invalid.
0x715	1813	0x98110715	ADSERR_DEVICE_CLIENTUNKNOWN	Notification client not registered.
0x716	1814	0x98110716	ADSERR_DEVICE_NOMOREHDLS	No further handle available.
0x717	1815	0x98110717	ADSERR_DEVICE_INVALIDWATCHSIZE	Notification size too large.
0x718	1816	0x98110718	ADSERR_DEVICE_NOTINIT	Device not initialized.
0x719	1817	0x98110719	ADSERR_DEVICE_TIMEOUT	Device has a timeout.
0x71A	1818	0x9811071A	ADSERR_DEVICE_NOINTERFACE	Interface query failed.
0x71B	1819	0x9811071B	ADSERR_DEVICE_INVALIDINTERFACE	Wrong interface requested.
0x71C	1820	0x9811071C	ADSERR_DEVICE_INVALIDCLSID	Class ID is invalid.
0x71D	1821	0x9811071D	ADSERR_DEVICE_INVALIDOBJID	Object ID is invalid.
0x71E	1822	0x9811071E	ADSERR_DEVICE_PENDING	Request pending.
0x71F	1823	0x9811071F	ADSERR_DEVICE_ABORTED	Request is aborted.
0x720	1824	0x98110720	ADSERR_DEVICE_WARNING	Signal warning.
0x721	1825	0x98110721	ADSERR_DEVICE_INVALIDARRAYIDX	Invalid array index.
0x722	1826	0x98110722	ADSERR_DEVICE_SYMBOLNOTACTIVE	Symbol not active.
0x723	1827	0x98110723	ADSERR_DEVICE_ACCESSDENIED	Access denied.
0x724	1828	0x98110724	ADSERR_DEVICE_LICENSENOTFOUND	Missing license.
0x725	1829	0x98110725	ADSERR_DEVICE_LICENSEEXPIRED	License expired.
0x726	1830	0x98110726	ADSERR_DEVICE_LICENSEEXCEEDED	License exceeded.
0x727	1831	0x98110727	ADSERR_DEVICE_LICENSEINVALID	Invalid license.
0x728	1832	0x98110728	ADSERR_DEVICE_LICENSESYSTEMID	License problem: System ID is invalid.
0x729	1833	0x98110729	ADSERR_DEVICE_LICENSENOTIMELIMIT	License not limited in time.
0x72A	1834	0x9811072A	ADSERR_DEVICE_LICENSEFUTUREISSUE	Licensing problem: time in the future.
0x72B	1835	0x9811072B	ADSERR_DEVICE_LICENSETIMETOLONG	License period too long.
0x72C	1836	0x9811072C	ADSERR_DEVICE_EXCEPTION	Exception at system startup.
0x72D	1837	0x9811072D	ADSERR_DEVICE_LICENSEDUPLICATED	License file read twice.
0x72E	1838	0x9811072E	ADSERR_DEVICE_SIGNATUREINVALID	Invalid signature.
0x72F	1839	0x9811072F	ADSERR_DEVICE_CERTIFICATEINVALID	Invalid certificate.
0x730	1840	0x98110730	ADSERR_DEVICE_LICENSEOEMNOTFOUND	Public key not known from OEM.
0x731	1841	0x98110731	ADSERR_DEVICE_LICENSERESTRICTED	License not valid for this system ID.
0x732	1842	0x98110732	ADSERR_DEVICE_LICENSEDEMODENIED	Demo license prohibited.
0x733	1843	0x98110733	ADSERR_DEVICE_INVALIDFNCID	Invalid function ID.
0x734	1844	0x98110734	ADSERR_DEVICE_OUTOFRANGE	Outside the valid range.
0x735	1845	0x98110735	ADSERR_DEVICE_INVALIDALIGNMENT	Invalid alignment.
0x736	1846	0x98110736	ADSERR_DEVICE_LICENSEPLATFORM	Invalid platform level.

Hex	Dec	HRESULT	Name	Description
0x737	1847	0x98110737	ADSERR_DEVICE_FORWARD_PL	Context – forward to passive level.
0x738	1848	0x98110738	ADSERR_DEVICE_FORWARD_DL	Context – forward to dispatch level.
0x739	1849	0x98110739	ADSERR_DEVICE_FORWARD_RT	Context – forward to real time.
0x740	1856	0x98110740	ADSERR_CLIENT_ERROR	Client error.
0x741	1857	0x98110741	ADSERR_CLIENT_INVALIDPARM	Service contains an invalid parameter.
0x742	1858	0x98110742	ADSERR_CLIENT_LISTEMPTY	Polling list is empty.
0x743	1859	0x98110743	ADSERR_CLIENT_VARUSED	Var connection already in use.
0x744	1860	0x98110744	ADSERR_CLIENT_DUPLINVOKEID	The called ID is already in use.
0x745	1861	0x98110745	ADSERR_CLIENT_SYNCTIMEOUT	Timeout has occurred – the remote terminal is not responding in the specified ADS timeout. The route setting of the remote terminal may be configured incorrectly.
0x746	1862	0x98110746	ADSERR_CLIENT_W32ERROR	Error in Win32 subsystem.
0x747	1863	0x98110747	ADSERR_CLIENT_TIMEOUTINVALID	Invalid client timeout value.
0x748	1864	0x98110748	ADSERR_CLIENT_PORTNOTOPEN	Port not open.
0x749	1865	0x98110749	ADSERR_CLIENT_NOAMSADDR	No AMS address.
0x750	1872	0x98110750	ADSERR_CLIENT_SYNCINTERNAL	Internal error in Ads sync.
0x751	1873	0x98110751	ADSERR_CLIENT_ADDHASH	Hash table overflow.
0x752	1874	0x98110752	ADSERR_CLIENT_REMOVEHASH	Key not found in the table.
0x753	1875	0x98110753	ADSERR_CLIENT_NOMORESYM	No symbols in the cache.
0x754	1876	0x98110754	ADSERR_CLIENT_SYNCRESINVALID	Invalid response received.
0x755	1877	0x98110755	ADSERR_CLIENT_SYNCPORTLOCKED	Sync Port is locked.
0x756	1878	0x98110756	ADSERR_CLIENT_REQUESTCANCELLED	The request was cancelled.

RTime error codes

Hex	Dec	HRESULT	Name	Description
0x1000	4096	0x98111000	RTERR_INTERNAL	Internal error in the real-time system.
0x1001	4097	0x98111001	RTERR_BADTIMERPERIODS	Timer value is not valid.
0x1002	4098	0x98111002	RTERR_INVALIDTASKPTR	Task pointer has the invalid value 0 (zero).
0x1003	4099	0x98111003	RTERR_INVALIDSTACKPTR	Stack pointer has the invalid value 0 (zero).
0x1004	4100	0x98111004	RTERR_PRIOEXISTS	The request task priority is already assigned.
0x1005	4101	0x98111005	RTERR_NOMORETCB	No free TCB (Task Control Block) available. The maximum number of TCBs is 64.
0x1006	4102	0x98111006	RTERR_NOMORESEMAS	No free semaphores available. The maximum number of semaphores is 64.
0x1007	4103	0x98111007	RTERR_NOMOREQUEUES	No free space available in the queue. The maximum number of positions in the queue is 64.
0x100D	4109	0x9811100D	RTERR_EXTIRQALREADYDEF	An external synchronization interrupt is already applied.
0x100E	4110	0x9811100E	RTERR_EXTIRQNOTDEF	No external sync interrupt applied.
0x100F	4111	0x9811100F	RTERR_EXTIRQINSTALLFAILED	Application of the external synchronization interrupt has failed.
0x1010	4112	0x98111010	RTERR_IRQLNOTLESSOREQUAL	Call of a service function in the wrong context
0x1017	4119	0x98111017	RTERR_VMXNOTSUPPORTED	Intel VT-x extension is not supported.
0x1018	4120	0x98111018	RTERR_VMXDISABLED	Intel VT-x extension is not enabled in the BIOS.
0x1019	4121	0x98111019	RTERR_VMXCONTROLSMISSING	Missing function in Intel VT-x extension.
0x101A	4122	0x9811101A	RTERR_VMXENABLEFAILS	Activation of Intel VT-x fails.

Specific positive HRESULT Return Codes:

HRESULT	Name	Description
0x0000_0000	S_OK	No error.
0x0000_0001	S_FALSE	No error. Example: successful processing, but with a negative or incomplete result.
0x0000_0203	S_PENDING	No error. Example: successful processing, but no result is available yet.
0x0000_0256	S_WATCHDOG_TIMEOUT	No error. Example: successful processing, but a timeout occurred.

TCP Winsock error codes

Hex	Dec	Name	Description
0x274C	10060	WSAETIMEDOUT	A connection timeout has occurred - error while establishing the connection, because the remote terminal did not respond properly after a certain period of time, or the established connection could not be maintained because the connected host did not respond.
0x274D	10061	WSAECONNREFUSED	Connection refused - no connection could be established because the target computer has explicitly rejected it. This error usually results from an attempt to connect to a service that is inactive on the external host, that is, a service for which no server application is running.
0x2751	10065	WSAEHOSTUNREACH	No route to host - a socket operation referred to an unavailable host.
More Winsock error codes: Win32 error codes			

8 Support and Service

Beckhoff and their partners around the world offer comprehensive support and service, making available fast and competent assistance with all questions related to Beckhoff products and system solutions.

Download finder

Our download finder contains all the files that we offer you for downloading. You will find application reports, technical documentation, technical drawings, configuration files and much more.

The downloads are available in various formats.

Beckhoff's branch offices and representatives

Please contact your Beckhoff branch office or representative for local support and service on Beckhoff products!

The addresses of Beckhoff's branch offices and representatives round the world can be found on our internet page: www.beckhoff.com

You will also find further documentation for Beckhoff components there.

Beckhoff Support

Support offers you comprehensive technical assistance, helping you not only with the application of individual Beckhoff products, but also with other, wide-ranging services:

- support
- design, programming and commissioning of complex automation systems
- and extensive training program for Beckhoff system components

Hotline:	+49 5246 963-157
e-mail:	support@beckhoff.com

Beckhoff Service

The Beckhoff Service Center supports you in all matters of after-sales service:

- on-site service
- repair service
- spare parts service
- hotline service

Hotline:	+49 5246 963-460
e-mail:	service@beckhoff.com

Beckhoff Headquarters

Beckhoff Automation GmbH & Co. KG

Huelshorstweg 20
33415 Verl
Germany

Phone:	+49 5246 963-0
e-mail:	info@beckhoff.com
web:	www.beckhoff.com

More Information:
www.beckhoff.com/te1000

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Germany
Phone: +49 5246 9630
info@beckhoff.com
www.beckhoff.com