LibSparkPlug
User Manual

MU214622 Rev. A

December 5, 2024

www.altus.com.br

According to current legislation in Brazil, the Consumer Defense Code, we are giving the following information to clients who use our products, regarding personal safety and premises.

The industrial automation equipment, manufactured by Altus, is strong and reliable due to the stringent quality control it is subjected to. However, any electronic industrial control equipment (programmable controllers, numerical commands, etc.) can damage machines or processes controlled by them when there are defective components and/or when a programming or installation error occurs. This can even put human lives at risk. The user should consider the possible consequences of the defects and should provide additional external installations for safety reasons. This concern is higher when in initial commissioning and testing.

The equipment manufactured by Altus does not directly expose the environment to hazards, since they do not issue any kind of pollutant during their use. However, concerning the disposal of equipment, it is important to point out that built-in electronics may contain materials which are harmful to nature when improperly discarded. Therefore, it is recommended that whenever discarding this type of product, it should be forwarded to recycling plants, which guarantee proper waste management.

It is essential to read and understand the product documentation, such as manuals and technical characteristics before its installation or use. The examples and figures presented in this document are solely for illustrative purposes. Due to possible upgrades and improvements that the products may present, Altus assumes no responsibility for the use of these examples and figures in real applications. They should only be used to assist user trainings and improve experience with the products and their features.

Altus warrants its equipment as described in General Conditions of Supply, attached to the commercial proposals.

Altus guarantees that their equipment works in accordance with the clear instructions contained in their manuals and/or technical characteristics, not guaranteeing the success of any particular type of application of the equipment.

Altus does not acknowledge any other guarantee, directly or implied, mainly when end customers are dealing with third-party suppliers. The requests for additional information about the supply, equipment features and/or any other Altus services must be made in writing form. Altus is not responsible for supplying information about its equipment without formal request. These products can use EtherCAT® technology (www.ethercat.org).

COPYRIGHTS

OPEN SOURCE SOFTWARE NOTICE

To obtain the source code under GPL, LGPL, MPL and other open source licenses, that is contained in this product, please contact opensource@altus.com.br. In addition to the source code, all referred license terms, warranty disclaimers and copyright notices may be disclosed under request.

Contents

1. Introduction . . . 1
   1.1. Documents Related to this Manual . . . 1
   1.2. Technical Support . . . 1
   1.3. Warning Messages Used in this Manual . . . 1
2. Sparkplug Features . . . 2
   2.1. Sparkplug Terminology in this Manual . . . 2
      2.1.1. MQTT Broker . . . 2
      2.1.2. EoN . . . 3
      2.1.3. Device . . . 3
      2.1.4. Metric . . . 3
      2.1.5. Host Application . . . 3
         2.1.5.1. Primary Host . . . 3
         2.1.5.2. Non-Primary Hosts . . . 3
   2.2. Compliance of Nexto Controllers with Sparkplug Specification . . . 4
   2.3. Buffering Capability of Nexto Controllers . . . 4
      2.3.1. Selective Buffering . . . 4
      2.3.2. Buffer Size . . . 4
3. Configuration . . . 6
   3.1. Software Versions . . . 6
   3.2. Library LibSparkplug . . . 6
      3.2.1. Function Block SPB_FB_SPARKPLUG . . . 7
         3.2.1.1. in_xEnable . . . 7
         3.2.1.2. in_sGroupId . . . 8
         3.2.1.3. in_sEonId . . . 8
         3.2.1.4. in_sPrimaryHostId . . . 8
         3.2.1.5. in_pEonMetrics . . . 8
         3.2.1.6. in_usiQtyEonMetrics . . . 9
         3.2.1.7. in_pDevices . . . 9
         3.2.1.8. in_usiQtyDevices . . . 10
         3.2.1.9. in_pStorageBuffer . . . 10
         3.2.1.10. in_uiSizeBuffer . . . 11
         3.2.1.11. in_stMqttParameters . . . 11
            3.2.1.11.1. sClientId . . . 11
            3.2.1.11.2. sHostName . . . 11
            3.2.1.11.3. sUser . . . 11
            3.2.1.11.4. sPass . . . 12
            3.2.1.11.5. uiPort . . . 12
            3.2.1.11.6. uiKeepAlive . . . 12

3.2.1.11.7. xEnableTLS . . . 12
3.2.1.11.8. sCertFilename . . . 13
3.2.1.11.9. sClientCert . . . 13
3.2.1.11.10. sClientKey . . . 13
3.2.1.12. in_timTimePeriodic . . . 13
3.2.1.13. out_xError . . . 13
3.2.1.14. out_xIsConnected . . . 13
3.2.1.15. out_xIsBuffering . . . 14
3.2.1.16. out_stStatus . . . 14
3.2.1.16.1. bDeviceNotFound . . . 14
3.2.1.16.2. bInvalidJson . . . 14
3.2.1.16.3. eProtobufStatus . . . 14
3.2.1.16.4. eMqttErrorCode . . . 15
3.2.1.17. out_eError . . . 15
3.2.2. Function Block SPB_FB_DEVICE . . . 15
3.2.2.1. in_xEnable . . . 16
3.2.2.2. in_sDeviceId . . . 16
3.2.2.3. in_usiMaxMetrics . . . 16
3.2.2.4. in_pMetrics . . . 16
3.2.2.5. out_eStatus . . . 17
3.2.2.6. out_eProtobufStatus . . . 17
3.2.3. Function Block SPB_FB_METRICS . . . 17
3.2.3.1. in_sName . . . 18
3.2.3.2. in_pValue . . . 18
3.2.3.3. in_usiMaxPayloadSize . . . 19
3.2.3.4. in_eDataType . . . 19
3.2.3.5. in_stPublishParameters . . . 19
3.2.3.5.1. ePubMode . . . 20
3.2.3.5.2. pBufferVariable . . . 20
3.2.3.5.3. rDeadBandValue . . . 20
3.2.3.5.4. usiTrunc . . . 21
3.2.3.6. in_xTrigger . . . 21
3.2.3.7. in_xEnableHist . . . 21
3.3. Diagnostics . . . 21
3.4. Example of Usage of Library LibSparkplug . . . 22
3.4.1. Variable Section of POU EoN . . . 22
3.4.2. Code Section of POU EoN . . . 25
3.5. Example of Authentication Configuration . . . 30
3.6. Example of TLS Security Configuration . . . 31
3.6.1. Adjust Clock and Keep them Synchronized . . . 32
3.6.2. Generate Certificates . . . 32
3.6.3. Install Certificate Files in the MQTT Broker . . . 34
3.6.4. Install Certificate Files in the EoN . . . 35
3.6.5. Edit the Configuration File of MQTT Broker Mosquitto . . . 35
3.6.6. Adjust the EoN Application . . . 36
3.7. Effects of Online Change . . . 36

1. Introduction

1.1. Documents Related to this Manual

1.2. Technical Support

For Altus Technical Support contact in São Leopoldo, RS, call +55 51 3589-9500. For further information regarding the Altus Technical Support existent on other places, see https://www.altus.com.br/en/ or send an email to altus@altus.com.br.

If the equipment is already installed, you must have the following information at the moment of support requesting:

- The model from the used equipments and the installed system configuration
- The product serial number
- The equipment revision and the executive software version, written on the tag fixed on the product's side
- CPU operation mode information, acquired through MasterTool IEC XE
- The application software content, acquired through MasterTool IEC XE
- Used programmer version

1.3. Warning Messages Used in this Manual

In this manual, the warning messages will be presented in the following formats and meanings:

DANGER

Reports potential hazard that, if not detected, may be harmful to people, materials, environment and production.

CAUTION

Reports configuration, application or installation details that must be taken into consideration to avoid any instance that may cause system failure and consequent impact.

ATTENTION

Identifies configuration, application and installation details aimed at achieving maximum operational performance of the system.

2. Sparkplug Features

ATTENTION

2.1. Sparkplug Terminology in this Manual

This section defines the main Sparkplug related terms used in this manual. These definitions may not be complete from the point of view of the Sparkplug specification. These definitions consider only a subset of applications that use Nexto Series controllers as EoNs (Edge Nodes).

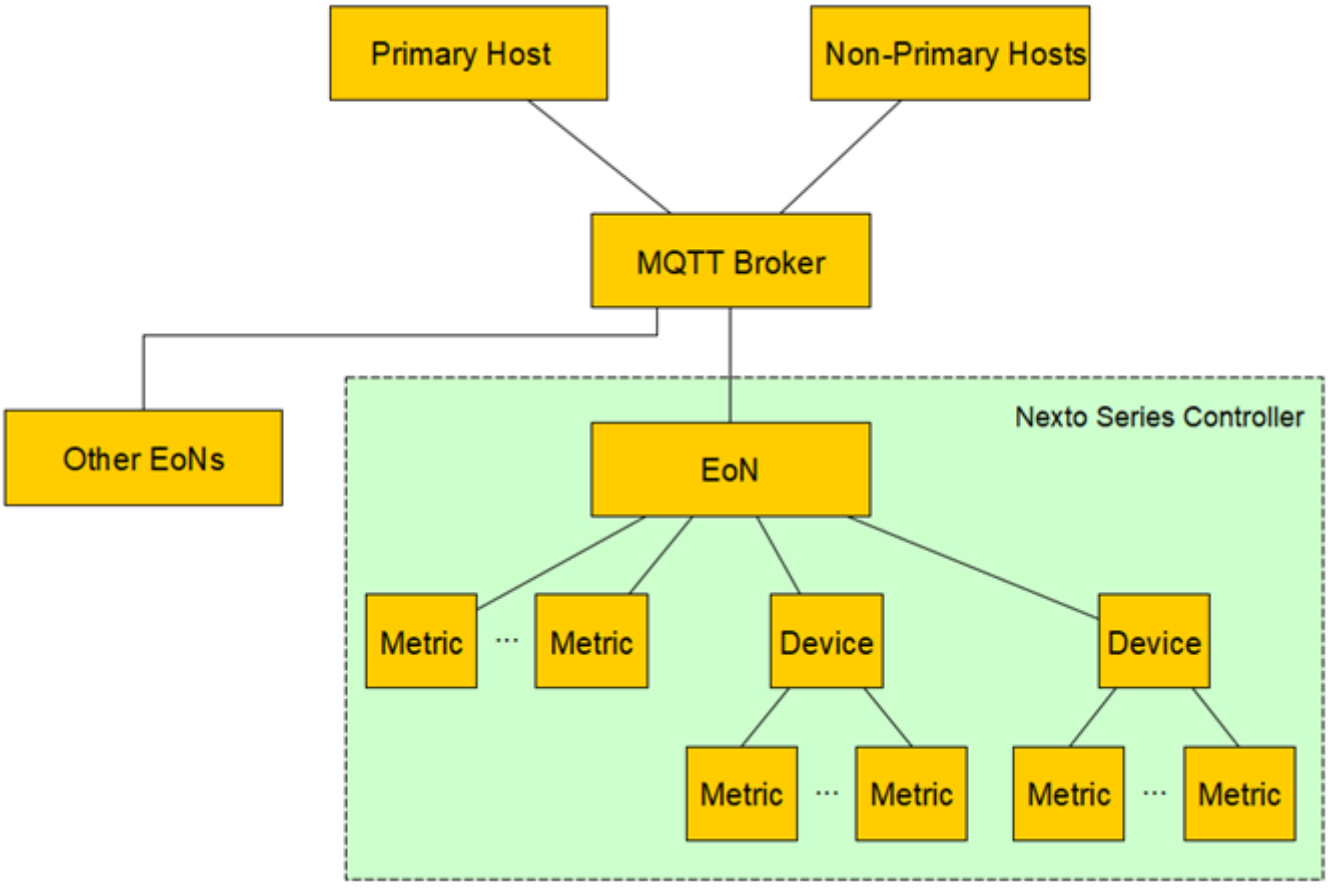


Figure 1: Main components of a Sparkplug infrastructure using a Nexto Series controller as EoN

The previous figure shows that an EoN implemented in a Nexto controller can contain several metrics and several devices. In addition, each of these devices can contain several metrics.

2.1.1. MQTT Broker

Also known as MQTT server.

Program or device that acts as an intermediary between MQTT clients which publish application messages and clients which have made subscriptions. MQTT enabled infrastructure requires that one or more MQTT brokers are present in the infrastructure.

2.1.2. EoN

Also known as Edge Node.

Any v3.1.1 or v5.0 compliant MQTT client application that manages an MQTT session and provides the physical and/or logical gateway functions required to participate in the Topic Namespace and Payload definitions described in the Sparkplug specification. The EoN is responsible for any local protocol interface to existing devices (PLCs, RTUs, Flow Computers, Sensors, etc.) and/or any local discrete I/O, and/or any logical internal process variables (PVs).

In a Nexto Series controller, it is possible to create several instances of an EoN (see section Function Block SPB_FB_SPARKPLUG). The user must judge if more than one instance makes sense for his application.

2.1.3. Device

Physical or logical device that makes sense in the context of a distributed Sparkplug application. Often times a Sparkplug Device will be a physical PLC, RTU, Flow Computer, Sensor, etc. However, a Sparkplug device could also represent a logical grouping of data points as makes sense for the specific Sparkplug Application being developed. For example, it could represent a set of data points across multiple PLCs that make up a logical device that makes sense within the context of that application.

Considering the specific case of a Nexto Controller, it is possible to create several instances of devices below instances of EoNs (see section Function Block SPB_FB_DEVICE). The following examples of devices make sense in a Nexto Controller:

- An I/O module in the local bus.
- An I/O module in a remote bus (bus expansion, remote I/O device, etc).
- A remote compact device (remote I/O device).
- A logic device (a group of logically related variables).

2.1.4. Metric

A Sparkplug metric typically includes a name, value, and timestamp.

Considering the specific case of a Nexto Controller, it is possible to create several instances of metrics below instances of EoNs or below instances of devices (see section Function Block SPB_FB_METRICS). The following examples of devices make sense in a Nexto Controller:

- Value of a logical variable (internal data).
- Value of a digital input or analog input.
- Value of a digital output or analog output (commands received from a host application).

2.1.5. Host Application

A Sparkplug Host Application is typically at a central location and primarily receives data from multiple EoNs. A host application may also send command messages to EoNs for writing to output metrics of EoNs and Devices below EoNs.

From the point of view of a Nexto controller as an EoN, two types of host application exist: a single Primary Host and any number of Non-Primary Hosts.

2.1.5.1. Primary Host

One single Primary Host must be configured in an EoN of a Nexto controller.

If the connection between this Primary Host and the MQTT server is lost, the EoN starts buffering data, as described in section Buffering Capability of Nexto Controllers.

While the EoN is buffering, it does not transmit data anymore to the MQTT broker, even if the EoN is still connected to the MQTT broker.

2.1.5.2. Non-Primary Hosts

Any number of Non-Primary Hosts can exist and receive data transmitted from an EoN configured in a Nexto Controller, as well send command messages to output metrics of this EoN.

It is not necessary to make any configuration in an EoN of a Nexto controller for declaring Non-Primary Hosts. Data and command retransmissions are entirely managed by the MQTT broker.

ATTENTION

While the EoN is buffering, it does not transmit data anymore to the MQTT broker, even if the EoN is still connected to the MQTT broker. In this situation, data displayed in a Non-Primary Host can be outdated and still indicate quality good.

2.2. Compliance of Nexto Controllers with Sparkplug Specification

A Nexto Series controller complies with version 3.0.0 of Sparkplug specification, and use the Sparkplug B encoding scheme (spBv1.0/# namespace).

It can play the role of an EoN in a Sparkplug infrastructure. Below this EoN, it is possible to configure devices and metrics. Below each device, it is also possible to configure metrics.

2.3. Buffering Capability of Nexto Controllers

A Nexto Series controller as an EoN has buffering capability.

When both the EoN and the Primary Host are connected to the MQTT broker, data is transmitted from the EoN to the MQTT broker, and then this data is relayed to the Primary Host and, also, to Non-Primary Hosts.

However, while the EoN or the Primary Host is not connected to the MQTT broker, the EoN buffers data in internal memory, instead of transmitting this data to the MQTT broker. A circular queue with a configurable storage capability is used for this purpose. If the storage capability of this circular queue overflows, older data is lost and overwritten with newer data.

Afterward, when the connection with MQTT broker is reestablished with both EoN and Primary Host, the buffered data is transmitted from the EoN to the MQTT broker, and then relayed to the Primary Host and Non-Primary Hosts, as "historical data".

With this feature, one can preserve data events that otherwise would be lost.

ATTENTION

Note that when the connection between the Primary Host and the MQTT broker is lost, the EoN will not transmit data to the MQTT broker. Therefore, data with wrong values (outdated) and quality good can appear in Non-Primary Hosts.

2.3.1. Selective Buffering

It is possible to select which metrics will be buffered (see input in_xEnableHist of Function Block SPB_FB_METRICS).

2.3.2. Buffer Size

The user can define the buffer size using inputs in_pStorageBuffer and in_uiSizeBuffer of Function Block SPB_FB_SPARKPLUG. The minimum buffer size must be 267, if buffering is enabled. It is possible to disable buffering by informing a size of zero bytes. The maximum buffer size is 65536 bytes.

The bigger the buffer size, the lower is the probability of losing historical data due to buffer overflow.

The following table shows the number of bytes allocated by each historical record of a metric according to its data type (data types are defined by input in_eDataType of Function Block SPB_FB_METRICS).

Sparkplug Data Type	IEC 611313 Data Type	Bytes per Record
Int8	SINT	13
Int16	INT	14
Int32	DINT	16
Int64	LINT	20
UInt8	USINT	13
UInt16	UINT	14
UInt32	UDINT	16
UInt64	ULINT	20

Sparkplug Data Type	IEC 611313 Data Type	Bytes per Record
Float	REAL	16
Double	LREAL	20
Boolean	BOOL	13
sString	STRING(n)	12 + n (maximum for a string with "n" bytes)

Table 1: Bytes allocated for each historical records in buffer

3. Configuration

This chapter describes the configuration of the Sparkplug communication with the Mastertool Programming System for a Nexto controller. The list of Nexto controllers that support Sparkplug appears in chapter Introduction.

3.1. Software Versions

The Sparkplug features described in this manual require the programming system Mastertool IEC XE (MT8500) version 3.70 or newer.

3.2. Library LibSparkplug

For configuring a Nexto controller as an EoN, it is necessary to add the library *LibSparkplug* in the *Library Manager*.

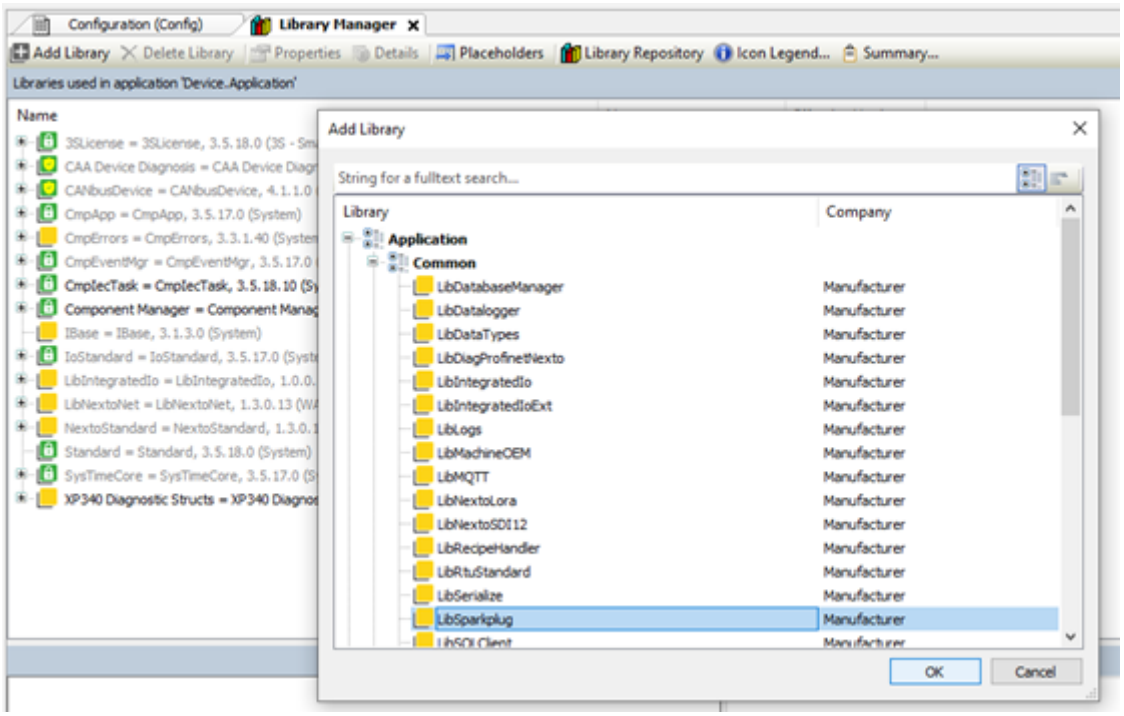


Figure 2: Adding library LibSparkplug in the Library Manager

The following figure shows the components of this library that the user must concern.

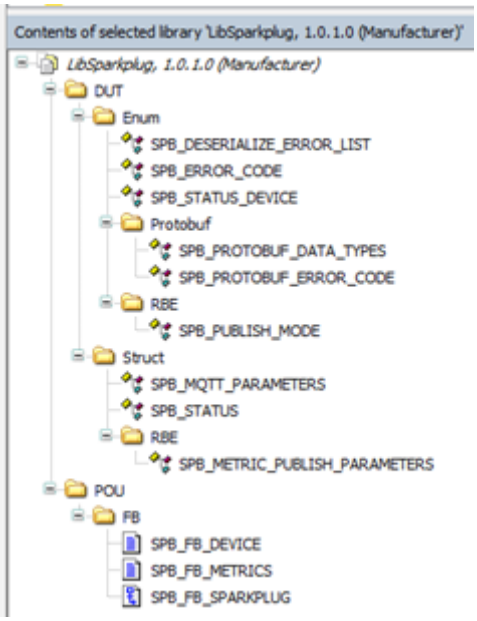


Figure 3: Components of library LibSparkplug

The following subsections describe the function blocks that make part of this library. The data types are described together with the function block descriptions.

3.2.1. Function Block SPB_FB_SPARKPLUG

This is the main function block of the library. It is possible to declare several instance of this FB. Each instance of this FB creates a different EoN in the PLC.

The following diagram shows the inputs and outputs of this FB.

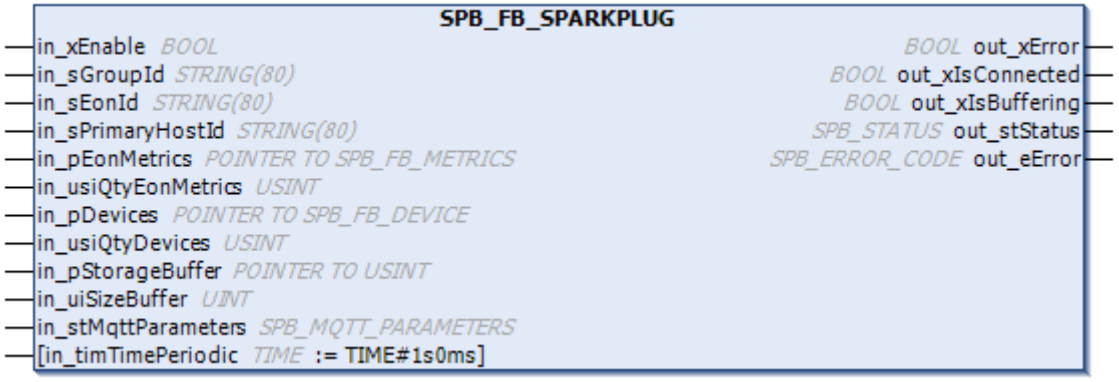


Figure 4: Function Block SPB_FB_SPARKPLUG

The following subsection describe the inputs and ouptus of this FB.

3.2.1.1. in_xEnable

- **Type:** BOOL

When this input changes from FALSE to TRUE, the EoN is created, the necessary processes for Sparkplug communication are started and the connection with the MQTT broker is established. Devices and metrics inside the EoN are also created.

When this input changes from TRUE to FALSE, the EoN disconnects from the MQTT broker.

Therefore, keep this input equal TRUE while Sparkplug communication with this EoN is desired.

3.2.1.2. in_sGroupId

- **Type:** STRING(80)

This string, with up to 80 characters, defines the name of the Sparkplug group to which the EoN belongs.

3.2.1.3. in_sEonId

- **Type:** STRING(80)

This string, with up to 80 characters, defines the name of the EoN. This name must be unique inside the Sparkplug group it belongs (see input parameter in_sGroupId).

3.2.1.4. in_sPrimaryHostId

- **Type:** STRING(80)

This string, with up to 80 characters, defines the name configured for the Primary Host.

The EoN must know the name of the Primary Host to monitor its communication status. This is necessary due to the buffering capability of Nexto Series controllers as an EoN, as described in the section Buffering Capability of Nexto Controllers.

3.2.1.5. in_pEonMetrics

- **Type:** POINTER TO SPB_FB_METRICS

This input must be initialized with the address of an array with type SPB_FB_METRICS. This array contains the user-defined metrics for the EoN.

Important notes:

- The first position of the array with metrics must be zero. Example for an array with 10 user-defined metrics:

```
afbEoN_Metrics : ARRAY [0 ... 9] OF LibSparkplug.SPB_FB_METRICS;
```

- Even if only one metric exists, an array must be used starting with index 0. Example for an array with only one user-defined metric:

```
afbEoN_Metric : ARRAY [0 ... 0] OF LibSparkplug.SPB_FB_METRICS;
```

- Use the ADR function to initialize this address correctly. Example:

```
ADR(afbEoN_Metrics[0])
```

- If this input's value is NULL, no user-defined metrics exist for the EoN. In this case, it is still possible to create metrics inside devices (see Function Block SPB_FB_DEVICE).

3.2.1.6. in_usiQtyEonMetrics

- **Type:** USINT

This input informs the quantity of user-defined metrics in the array pointed by the input in_pEonMetrics.

Notes:

- The minimum number of metrics below the EoN is 0.
- The maximum number of metrics below the EoN is 253.
- The value of this input should be 10, considering the following example of an array with 10 user defined metrics:

```
afbEoN_Metrics : ARRAY [0 ... 9] OF LibSparkplug.SPB_FB_METRICS;
```

- Example of alternative way for defining this input:

```
TO_USINT(SIZEOF(afbEoN_Metrics) / SIZEOF(LibSparkplug.SPB_FB_METRICS));
```

- The value 0 means that no user defined metrics exist for the EoN.

> **ATTENTION**
>
> If the value of in_usiQtyEonMetrics is assigned with a value different from the size of the array pointed by the input in_pEonMetrics, a malfunction (exception) may occur in the program execution.

3.2.1.7. in_pDevices

- **Type:** POINTER TO SPB_FB_DEVICE

This input must be initialized with the address of the first position of an array with type SPB_FB_DEVICE. This array contains the user-defined instances of devices for the EoN.

Notes:

- The first position of the array with devices must be zero. Example for an array with 10 user-defined devices:

```
afbEoN_Devices : ARRAY [0 ... 9] OF LibSparkplug.SPB_FB_DEVICE;
```

- Even if only one device exists, an array must be used starting with index 0. Example for an array with only one user-defined device:

```
afbEoN_Device : ARRAY [0 ... 0] OF LibSparkplug.SPB_FB_DEVICE;
```

- Use the ADR function to initialize this address correctly. Example:

```
ADR(afbEoN_Devices[0])
```

- If this input's value is NULL, no user-defined devices exist for the EoN.

3.2.1.8. in_usiQtyDevices

- **Type:** USINT

This input informs the quantity of user-defined devices in the array pointed by the input in_pDevices described in the previous section.

Notes:

- The minimum number of devices per EoN is 0 (no devices below the EoN).
- The maximum number of devices per EoN is 255.
- The value of this input should be 10, considering the following example of an array with 10 user defined metrics:

```
afbEoN_Devices : ARRAY [0 ... 9] OF LibSparkplug.SPB_FB_DEVICE;
```

- Example of alternative way for defining this input:

```
TO_USINT(SIZEOF(afbEoN_Devices) / SIZEOF(LibSparkplug.SPB_FB_DEVICE));
```

ATTENTION

If the value of in_usiQtyDevices is assigned with a value different from the size of the array pointed by the input in_pDevices, a malfunction (exception) may occur in the program execution.

3.2.1.9. in_pStorageBuffer

- **Type:** POINTER TO USINT

This input must be initialized with the address of the first position of an array of bytes (type USINT) used for the buffering function (see section Buffering Capability of Nexto Controllers).

Notes:

- The first position of the array must be zero. Example for an array with 1000 bytes:

```
ausiBuffer : ARRAY [0 ... 999] OF USINT;
```

- Use the ADR function to initialize this address correctly. Example:

```
ADR(ausiBuffer[0])
```

- If the value of this input is NULL, this means that no buffering is provided for the EoN.

3.2.1.10. in_uiSizeBuffer

- **Type:** UINT

This input informs the quantity of bytes in the array pointed by the input in_pStorageBuffer described in previous section.

Notes:

- The minimum buffer size is 267. If the buffer has less than 267 bytes, the output out_xError will return the value *STORAGE_ERROR* right after the first attempt to insert a metric in the buffer.
- The maximum buffer size is 65536 bytes.
- The value of this input should be 1000, considering the following example of an array with 1000 bytes:

```
ausiBuffer : ARRAY [0 ... 999] OF USINT;
```

- Example of alternative way for defining this input:

```
TO_UINT(SIZEOF(ausiBuffer) / SIZEOF(USINT));
```

- The value 0 means that no buffering is provided for the EoN.

> **ATTENTION**
>
> If the value of in_uiSizeBuffer is assigned with a value different from the size of the array pointed by the input in_pStorageBuffer, a malfunction (exception) may occur in the program execution.

3.2.1.11. in_stMqttParameters

- **Type:** SPB_MQTT_PARAMETERS

This input is a structure that configures the connection with the MQTT broker.

The following subsections define the fields of this structure.

3.2.1.11.1. sClientId

- **Type:** STRING(80)

This string is an optional client ID for the EoN. If this string is empty, a random ID is created.

3.2.1.11.2. sHostName

- **Type:** STRING(80)

This string must contain an URL or an IP address for the MQTT broker.

Example: '192.168.201.141'.

3.2.1.11.3. sUser

- **Type:** STRING(80)

This string must contain an user name for authentication. If authentication is not required for connecting to the MQTT broker, this string must be empty.

> **ATTENTION**
>
> For using this feature, it is also necessary to make appropriate configurations in the MQTT broker. Pleaser read the documentation of the MQTT broker. An example is given in section Example of Authentication Configuration.

3.2.1.11.4. sPass

- **Type:** STRING(80)

This string must contain a password for authentication. If authentication is not required for connecting to the MQTT broker, this string must be empty.

> **ATTENTION**
>
> For using this feature, it is also necessary to make appropriate configurations in the MQTT broker. Pleaser read the documentation of the MQTT broker. An example is given in section Example of Authentication Configuration.

3.2.1.11.5. uiPort

- **Type:** UINT
- **Default:** 1883

This input defines the TCP port used for connection with the MQTT broker. Two options can be used:

- **1883:** without encryption
- **8883:** with TLS encryption

The TCP port must be configured in accordance with the MQTT broker configuration.

> **ATTENTION**
>
> It may be necessary to configure the firewall of computer running the MQTT broker for accepting connection in this port.

3.2.1.11.6. uiKeepAlive

- **Type:** UINT
- **Default:** 60 seconds

This input defines the interval in seconds between keep-alive messages sent from the EoN to the MQTT broker for checking the integrity of the connection between them.

The smaller this interval, the faster the EoN and the MQTT broker will detect a disconnection. In the other hand, a smaller interval causes more traffic for sending more frequent keep alive messages.

A keep alive interval must be configured in all types of MQTT clients connected to an MQTT broker (EoNs, hosts, etc). When the MQTT broker detects that some MQTT client is disconnected, it informs this disconnection to other MQTT clients, for taking appropriate actions like the following:

- If a host is informed by the MQTT broker about disconnection of an EoN, this host must change to bad the quality of metrics coming from this EoN.
- If a Nexto controller EoN is informed about disconnection of the primary host, this EoN must switch to the "buffering" mode (see section out_xIsBuffering).

3.2.1.11.7. xEnableTLS

- Type: BOOL
- Default: FALSE

This input must be TRUE if the connection with the MQTT broker requires TLS encryption (versions 1.0, 1.1 or 1.2). In this case, the MQTT broker must implement TLS encryption using a CA certificate file.

More details for setting up a connection with TLS encryption are described in section Example of Authentication Configuration.

3.2.1.11.8. *sCertFilename*

- **Type:** STRING(80)

This string must contain the name of the CA certificate file provided by the MQTT broker. This string must be initialized only when the previous input xEnableTLS is TRUE.

More details for setting up a connection with TLS encryption are described in section Example of Authentication Configuration.

3.2.1.11.9. *sClientCert*

- **Type:** STRING(80)

This string must contain the name of the EoN (MQTT client) certificate file, created from the CA certificate file provided by the MQTT broker. This string must be initialized only when the previous input xEnableTLS is TRUE.

More details for setting up a connection with TLS encryption are described in section Example of Authentication Configuration.

3.2.1.11.10. *sClientKey*

- **Type:** STRING(80)

This string must contain the name of the EoN (MQTT client) key that makes part of the client certificate. This string must be initialized only when the previous input xEnableTLS is TRUE.

More details for setting up a connection with TLS encryption are described in section .

3.2.1.12. in_timTimePeriodic

- **Type:** TIME
- **Default:** T#1S

This input defines the period used for sampling changes in variables of metrics reported by exception or by deadand (section ePubMode describes how to configure the report method: report by exception, deadband or trigger).

The minimum period is 1 second.

- If the period is 0, changes are not detected using methods report by exception or deadband.
- If the period is bigger than 0, but lower than 1 second, a period of 1 second is used.

> **ATTENTION**
>
> The real sampling period can exceed in_timTimePeriodic in some situations:
> 1) When the CPU is overloaded (too many metrics for this CPU model).
> 2) When the EoN publishes metrics. It takes some time between consecutive samples to publish metrics.
> 3) If the MainTask interval is too high.

3.2.1.13. out_xError

- **Type:** BOOL

The value TRUE in this output indicates that some error was detected by the function block.

To recover from this error, besides removing the error's cause, it is necessary to disable and then enable again the function block, using the input in_xEnable.

Other diagnostics described in the next sections explain the error's cause.

3.2.1.14. out_xIsConnected

- **Type:** BOOL

The value TRUE in this output indicates that the EoN is connected to the MQTT broker.

Note that it may take some time to update correctly this output. This happens because some types of disconnection are detected using a keep-alive interval. See section uiKeepAlive.

3.2.1.15. out_xIsBuffering

- **Type:** BOOL

The value TRUE in this output indicates that the EoN is buffering metrics in internal storage, because some problem prevents the transmission of these metrics to the primary host. After this problem is solved, the metrics buffered in the internal storage will be transmitted as historical values (see section Buffering Capability of Nexto Controllers).

The following four situations can occur analysing out_xIsBuffering and out_xIsConnected together:

1. **out_xIsBuffering = FALSE and out_xIsConnected = FALSE:** this combination is only expected during initialization phase of the function block.
2. **out_xIsBuffering = FALSE and out_xIsConnected = TRUE:** this is the normal situation when buffering is not necessary, because both the EoN and the primary host are connected to the MQTT broker.
3. **out_xIsBuffering = TRUE and out_xIsConnected = FALSE:** the EoN is not conneted to the MQTT broker.
4. **out_xIsBuffering = TRUE and out_xIsConnected = TRUE:** the EoN is conneted to the MQTT broker, but the primary host is not connected to the MQTT broker.

Note that it may take some time for updating correctly the outputs out_xIsConnected and out_xIsBuffering. This happens because some types of disconnection are detected using a keep alive interval. The keep alive interval is configured for every MQTT client (EoN and hosts). See section uiKeepAlive.

> **ATTENTION**
>
> While out_xIsBuffering = TRUE and out_xIsConnected = TRUE, the EoN does not publish metrics to the MQTT broker. Therefore, a non-primary host may have metrics with outdated values but with quality good.

3.2.1.16. out_stStatus

- **Type:** SPB_STATUS

This output is a struct that shows status information about the function block and Sparkplug communication.

The following subitems describe the fields of this structure.

3.2.1.16.1. bDeviceNotFound

- **Type:** BIT

A command received from a host application has specified a device not configured in this EoN.

3.2.1.16.2. bInvalidJson

- **Type:** BIT

JSON published by the host application is invalid.

3.2.1.16.3. eProtobufStatus

- **Type:** SPB_PROTOBUF_ERROR_CODE

This enumeration defines an error code related to the "Google protocol buffer" applied in the Sparkplug B specification.

The following values are defined in this enumeration:

- **NONE:** no error.
- **METRIC_NOT_FOUND:** command received from a host application specified a metric not configured in this EoN, or the command does not included the alias field that identifies the metric.
- **PROTOBUF_PAYLOAD_INVALID:** command received from a host application has encoding problems.
- **PROTOBUF_STRING_OVERFLOW:** payload of string received in a command from a host application exceeds the payload configured for this string in a metric.
- **FIELD_TAG_UNKNOWN:** payload received from a host application has an unknown or not supported field.
- **METRIC_LIST_MISSING:** internal error of function block. Contact the suport team of ALTUS if such a diagnostic occurs.

3.2.1.16.4. *eMqttErrorCode*

- **Type:** LibMQTT.MQTT_ERR_CODE

This enumeration defines an error code related to the communication with the MQTT broker.

The value MQTT_NO_ERROR indicates that the MQTT communication is good.

The following values are defined in this enumeration:

- MQTT_CONNECTION_PENDING
- MQTT_NO_ERROR
- MQTT_ERROR_NO_MEMORY
- MQTT_ERROR_PROTOCOL_COMM
- MQTT_ERROR_INVALID_PARAM
- MQTT_ERROR_NO_CONNECTION
- MQTT_ERROR_CONNECTION_REFUSED
- MQTT_ERROR_NOT_FOUND
- MQTT_ERROR_CONNECTION_LOST
- MQTT_ERROR_TLS_CRYPTO
- MQTT_ERROR_PLAYLOAD_SIZE
- MQTT_ERROR_THREAD_NOT_SUPPORTED
- MQTT_ERROR_AUTHORIZATION
- MQTT_ERROR_ACL_DENIED
- MQTT_ERROR_UNKNOWN
- MQTT_ERROR_SYSTEM_CALL
- MQTT_ERROR_EAI
- MQTT_ERROR_PROXY

3.2.1.17. out_eError

- **Type:** SPB_ERR_CODE

This output is an enumeration that defines an error code related to the Sparkplug library.

The following values are defined in this enumeration:

- **NONE:** no error.
- **METRICS_TO_EON_OVERFLOW:** number of metrics created by the user below the EoN exceed 253.
- **DEVICE_MISS_METRIC_LIST:** some device does not have a metrics list. Any device must have at least one metric.
- **METRIC_PAYLOAD_SIZE_AS_ZERO:** some metric created by the user informs payload size equal zero. This error code applies for all metrics (created below the EoN and created below devices).
- **STORAGE_ERROR:** an error was detected in the storage buffer.

3.2.2. Function Block SPB_FB_DEVICE

Below the EoN created using an instance of Function Block SPB_FB_SPARKPLUG, it is possible to create several devices. This can be done using an array of instances of the Function Block SPB_FB_DEVICE (see input in_pDevices of Function Block SPB_FB_SPARKPLUG).

The following diagram shows the inputs and outputs of the Function Block SPB_FB_DEVICE.

Figure 5: Function Block SPB_FB_DEVICE

The following subsection describe the inputs and ouptus of this FB.

3.2.2.1. in_xEnable

- **Type:** BOOL

When this input changes from FALSE to TRUE, this indicates for the EoN that the device and their metrics became active. When this input changes from TRUE to FALSE, this indicates for the EoN that the device and their metrics became inactive. Therefore, keep this input equal TRUE while Sparkplug communication with this device is desired.

3.2.2.2. in_sDeviceId

- **Type:** STRING(80)

This string, with up to 80 characters, defines the name of the device inside the EoN. This name must be unique inside the EoN it belongs.

3.2.2.3. in_usiMaxMetrics

- **Type:** USINT

This input informs the quantity of user-defined metrics for the device, in the array pointed by the input in_pMetrics.

Notes:

- The minimum number of metrics per device is 1 (it makes no sense to create a device without metrics).
- The maximum number of metrics per device is 253.
- The value of this input should be 10, considering the following example of an array with 10 user-defined metrics:

```
afbDev0_Metrics : ARRAY [0 ... 9] OF LibSparkplug.SPB_FB_METRICS;
```

- Example of alternative way for defining this input:

```
TO_USINT(SIZEOF(afbDev0_Metrics) / SIZEOF(LibSparkplug.SPB_FB_METRICS));
```

ATTENTION

If the value of in_usiMaxMetrics is assigned with a value different from the size of the array pointed by the input in_pMetrics, a malfunction (exception) may occur in the program execution.

3.2.2.4. in_pMetrics

- **Type:** POINTER TO SPB_FB_METRICS

This input must be initialized with the address of the first position of an array with type SPB_FB_METRICS. This array contains the user-defined metrics for the device.

Notes:

- The first position of the array must be zero. Example for an array with 10 user defined metrics:

```
afbDev0_Metrics : ARRAY [0 ... 9] OF LibSparkplug.SPB_FB_METRICS;
```

- Even if only one metric exists, an array must be used starting with index 0. Example for an array with only one user-defined metric:

```
afbDev0_Metric : ARRAY [0 ... 0] OF LibSparkplug.SPB_FB_METRICS;
```

- Use the ADR function to initialize this address correctly. Example:

```
ADR(afbDev0_Metrics[0])
```

- If the value of this input is NULL, this means that no user defined metrics exist for the device. This should never occur, because it makes no sense to create a device without metrics.

3.2.2.5. out_eStatus

- **Type:** SPB_STATUS_DEVICE

This output is an enumeration that informs the status for the device and all their metrics.

The following values are defined in this enumeration:

- **Device_Dead:** the device and all their metrics are disabled.
- **Device_Alive:** the device and all their metrics are enabled.

3.2.2.6. out_eProtobufStatus

- **Type:** SPB_PROTOBUF_ERROR_CODE

This is an enumeration that defines an error code related to the "google protocol buffer" used in the Sparkplug B specification.

The following values are defined in this enumeration:

- **NONE:** no error.
- **METRIC_NOT_FOUND:** command received from a host application specified a metric not configured in this device.
- **PROTOBUF_PAYLOAD_INVALID:** command received from a host application has encoding problems.
- **DATA_TYPE_METRIC_IS_DIFFERENT:** the type of metric received in a command from a host application is different from the one informed by device on birth.
- **FIELD_TAG_UNKNOWN:** payload received from a host application has an unknown or not supported field.
- **METRIC_LIST_MISSING:** internal error of function block. Contact the support team if such a diagnostic occurs.

3.2.3. Function Block SPB_FB_METRICS

Below the EoN (instance of Function Block SPB_FB_SPARKPLUG) it is possible to create several metrics (see inputs in_pEonMetrics and in_usiQtyEonMetrics of Function Block SPB_FB_SPARKPLUG).

Furthermore, below a device (instance of FB SPB_FB_DEVICE) it is possible to create several metrics (see inputs in_pMetrics and in_usiMaxMetrics of Function Block SPB_FB_DEVICE).

For creating metrics below an EoN or a device, use an array of instances of the Function Block SPB_FB_METRICS. The following diagram shows the inputs of this FB.

Figure 6: Function Block SPB_FB_METRICS

The following subsection describe the inputs of this FB.

3.2.3.1. in_sName

- **Type:** STRING(80)
- **Default value:** "" (empty string)

This string, with up to 80 characters, defines the name of the metric in the EoN or in the device. This name must be unique inside the EoN or device it belongs.

3.2.3.2. in_pValue

- **Type:** POINTER TO USINT
- **Default value:** 0 (NULL)

This input informs the address of the variable reported by the metric. Use the function ADR for passing the address of this variable.

Example 1:

- Example of declaration of a variable with type INT:

```
iVar : INT;
```

- Example of pointer initialization for this variable:

```
ADR(iVar)
```

Example 2:

- Example of declaration of a variable with type STRING:

```
sVar : STRING(20);
```

- Example of pointer initialization for this variable:

```
ADR(sVar)
```

ATTENTION

A variable with type STRING must not exceed 254 bytes, so the maximum type is STRING(254).

3.2.3.3. in_usiMaxPayloadSize

- **Type:** USINT

This input informs the size (in bytes) of the variable to be reported by the metric. For setting this input correctly, it is highly recommended to use the construction "TO_USINT(SIZEOF(<type name>))".

For instance, if a variable has type INT, use the following function:

```
TO_USINT(SIZEOF(INT))
```

For instance, if a variable has type STRING(20), use the following function:

```
TO_USINT(SIZEOF(STRING(20)))
```

ATTENTION

A variable with type STRING must not exceed 254 bytes, so the maximum type is STRING(254). When the limit type STRING(254) is used, the expression TO_USINT(SIZEOF(STRING(254))) returns the value 255 (254 + 1). The additional byte is reserved for a "zero" used for indicating the string termination.

3.2.3.4. in_eDataType

- **Type:** SPB_PROTOBUF_DATA_TYPES

This input is an enumeration that informs the data type of the variable to be reported by the metric.

The following values are defined in this enumeration, and the equivalent types in the PLC languages (IEC 61131-3).

- **Int8:** same as SINT
- **Int16:** same as INT
- **Int32:** same as DINT
- **Int64:** same as LINT
- **UInt8:** same as USINT
- **UInt16:** same as UINT
- **UInt32:** same as UDINT
- **UInt64:** same as ULINT
- **Float:** same as REAL
- **Double:** same as LREAL
- **Boolean:** same as BOOL
- **sString:** same as STRING
- **Unknown:** reserved for future expansion

3.2.3.5. in_stPublishParameters

- **Type:** SPB_METRIC_PUBLISH_PARAMETERS

This input is a structure which fields are described in the following subsections.

3.2.3.5.1. ePubMode

- **Type:** SPB_PUBLISH_MODE
- **Default:** ONLY_TRIGGER

This field is an enumeration that enables to select between three publish modes for defining when the metric is reported to the MQTT broker.

- **ONLY_TRIGGER:** the metric is only published when a rising edge is detected in input in_xTrigger of Function Block SPB_FB_METRICS.
- **RBE:** the metric is pubished when data changes. Very small changes can be disregarded using the field usiTrunc of the input in_stPublishParameters of Function Block SPB_FB_METRICS.
- **DEAD_BAND_ABSOLUTE:** the metric is published when data changes by a value bigger than an absolute deadband defined by the field rDeadBandValue of the input in_stPublishParameters of Function Block SPB_FB_METRICS.

Regardless of the selected publish mode, the metric is always reported to the MQTT broker when a rising edge is detected in input in_xTrigger of Function Block SPB_FB_METRICS.

ATTENTION

The publish mode DEAD_BAND_ABSOLUTE makes no sense and must not be used for the data types Boolean (BOOL) and sString (STRING). If this is attempted, the metric will not be reported based on deadband. For these data types, please use ONLY_TRIGGER or RBE.

3.2.3.5.2. pBufferVariable

- **Type:** POINTER TO USINT

This field is a pointer to an auxiliary variable necessary when the publish mode selected by the field ePubMode of input in_stPublishParameters is RBE or DEAD_BAND_ABSOLUTE. If the publish mode is ONLY_TRIGGER, this pointer can be NULL.

The auxiliary variable must have the same type of variable reported by the metric, pointed by input in_pValue of Function Block SPB_FB_METRICS.

ATTENTION

If the type of the auxiliary variable is different from the type of variable reported by the metric, a malfunction (exception) may occur during the program execution.

3.2.3.5.3. rDeadBandValue

- **Type:** REAL

This field will only be considered when the following conditions are true:

- The publish mode selected by the field ePubMode of input in_stPublishParameters is DEAD_BAND_ABSOLUTE.
- The data type selected by the input in_eDataType is one of the following:
  - Int8 (SINT)
  - Int16 (INT)
  - Int32 (DINT)
  - Int64 (LINT)
  - UInt8 (USINT)
  - UInt16 (UINT)
  - UInt32 (UDINT)
  - UInt64 (ULINT)
  - Float (REAL)
  - Double (LREAL)

This field defines the value of an absolute deadband that avoids publishing metrics with small changes when publish mode is DEAD_BAND_ABSOLUTE. The metric will only be published if the change in the variable reported by the metric is bigger than rDeadBandValue.

3.2.3.5.4. *usiTrunc*

- **Type:** USINT

This field will only be considered when the following conditions are true:

- The publish mode selected by the field ePubMode of input in_stPublishParameters is RBE.
- The data type selected by the input in_eDataType is one of the following:
  - Float (REAL)
  - Double (LREAL)

This field informs the number of digits after the decimal point that will be considered for evaluating if the metric will be published. For instance, considering that usiTrunc = 2:

- If the value of the metric changes from 5.1 to 5.12, it will be published.
- If the value of the metric changes from 5.12 to 5.123, it will not be published, because the change occurred in the third digit after the decimal point.

This field avoids the transmission of very small changes in floating point metrics when publish mode is RBE.

> **ATTENTION**
>
> The maximum value considered for usiTrunc is 5. If a bigger value of usiTrunc is informed by the user, it will be managed as if usiTrunc = 5.

3.2.3.6. in_xTrigger

- **Type:** BOOL
- **Default:** FALSE

When this input changes from FALSE to TRUE, the metric is reported to the MQTT broker. This happens even when the publish mode is RBE or DEAD_BAND_ABSOLUTE.

This input is the only way for reporting the metric when the publish mode is ONLY_TRIGGER.

3.2.3.7. in_xEnableHist

- **Type:** BOOL
- **Default:** FALSE

When this input is TRUE, the metric will be stored in the storage buffer while the EoN or the primary host are disconnected from the MQTT broker (see section Buffering Capability of Nexto Controllers).

Set this input to FALSE if buffering this metric is not necessary.

3.3. Diagnostics

All diagnostics related to Sparkplug communication are outputs of the function blocks described in section Library Lib-Sparkplug of the current chapter:

- Diagnostic outputs of Function Block SPB_FB_SPARKPLUG:
  - out_xError
  - out_xIsConnected
  - out_xIsBuffering
  - out_stStatus
  - out_eError
- Diagnostic outputs of Function Block SPB_FB_DEVICE:
  - out_eStatus
  - out_eProtobufStatus
- Diagnostic outputs of Function Block SPB_FB_METRICS: none

3.4. Example of Usage of Library LibSparkplug

This section shows an example for configuring an EoN with the following features:

- Three metrics directly below the EoN.
- Two devices below the EoN:
  - Device 0 with two metrics.
  - Device 1 with one metric.

Initially, the library LibSparkPlug was added in the Library Manager.

After this, the variables and code for configuring the EoN were inserted in a POU called Eon (POU of type PROGRAM developed with ST language). The POU EoN was called inside the POU UserPrg.

Comments in the following subsections explain details about this example.

3.4.1. Variable Section of POU EoN

```
PROGRAM EoN
VAR
// Variables for instance of EoN. Other inputs of instance passed directly in the code section.
fbEoN : LibSparkplug.SPB_FB_SPARKPLUG;                          // Instance of the main FB for the EoN
xEnableEon : BOOL := TRUE;                                      // Variable for enabling the main FB for the EoN
afbEonMetrics : ARRAY[0..2] OF LibSparkplug.SPB_FB_METRICS;     // Array with FBs for three metrics (0 .. 2) below the EoN
afbDevice : ARRAY[0..1] OF LibSparkplug.SPB_FB_DEVICE;          // Array with FBs for two devices (0 .. 1) below the EoN
abStorageBuffer : ARRAY[0..4999] OF USINT;                      // Array with the storage buffer with 5000 bytes
stMqttParameter : LibSparkplug.SPB_MQTT_PARAMETERS;             // Structure with parameters for connection with the MQTT broker
xError : BOOL;                                                  // Output for indicating presence of an error
xIsConnected : BOOL;                                            // Output for indicating connection to the MQTT borker
xIsBuffering : BOOL;                                            // Output for indicating buffering mode
stStatus : LibSparkplug.SPB_STATUS;                             // Structure with diagnostics about Sparkplug communication
eError : LibSparkplug.SPB_ERROR_CODE;                           // Enumeration for indicating an error code detected by this FB

// Variables for instance of Device 0 below the EoN. Other inputs of instance passed directly in the code section.
xEnableDev0 : BOOL := TRUE;                                     // Variable for enabling the FB for Device 0
afbDev0Metrics : ARRAY[0..1] OF LibSparkplug.SPB_FB_METRICS;    // Array with FBs for the two metrics (0 .. 1) below Device 0
eStatusDev0 : LibSparkplug.SPB_STATUS_DEVICE;                   // Status for Device 0
eProtobufStatusDev0 : LibSparkplug.SPB_PROTOBUF_ERROR_CODE;     // Protobuf error codes for Device 0

// Variables for instance of Device 1 below the EoN. Other inputs of instance passed directly in the code section.
xEnableDev1 : BOOL := TRUE;                                     // Variable for enabling the FB for Device 1
afbDev1Metrics : ARRAY[0..0] OF LibSparkplug.SPB_FB_METRICS;    // Array with FB for the one metric (0 .. 0) below Device 1
eStatusDev1 : LibSparkplug.SPB_STATUS_DEVICE;                   // Status for Device 1
eProtobufStatusDev1 : LibSparkplug.SPB_PROTOBUF_ERROR_CODE;     // Protobuf error codes for Device 1

// Variables for Metric 0 directly below the EoN
rVar_Metric0_EoN : REAL;                // Variable reported by this metric
rVar_Metric0_EoN_aux : REAL;            // Auxiliary variable for reporting this metric by exception or deadband

// Variables for Metric 1 directly below the EoN
iVar_Metric1_EoN : INT;                 // Variable reported by this metric
iVar_Metric1_EoN_aux : INT;             // Auxiliary variable for reporting this metric by exception or deadband

// Variables for Metric 2 directly below the EoN
xVar_Metric2_EoN : BOOL;                // Variable reported by this metric
xVar_Metric2_EoN_aux : BOOL;            // Auxiliary variable for reporting this metric by exception or deadband

// Variables for Metric 0 of Device 0
lrVar_Metric0_Dev0 : LREAL;             // Variable reported by this metric
lrVar_Metric0_Dev0_aux : LREAL;         // Auxiliary variable for reporting this metric by exception or deadband

// Variables for Metric 1 of Device 0
uiVar_Metric1_Dev0 : UINT;              // Variable reported by this metric
uiVar_Metric1_Dev0_aux : UINT;          // Auxiliary variable for reporting this metric by exception or deadband

// Variables for Metric 0 of Device 1
liVar_Metric0_Dev1 : LINT;              // Variable reported by this metric
liVar_Metric0_Dev1_aux : LINT;          // Auxiliary variable for reporting this metric by exception or deadband
END_VAR
```

Figure 7: Variables of POU EoN

The following text box is a copy of the variable section shown in the previous figure. It is intended to copy and paste this example, should you want to test it.

```
PROGRAM EoN
VAR
// Variables for instance of EoN. Other inputs of instance passed directly in
   the code section.
fbEoN : LibSparkplug.SPB_FB_SPARKPLUG;              // Instance of the main FB
   for the EoN
xEnableEon : BOOL := TRUE;                     // Variable for enabling the main
   FB for the EoN
afbEonMetrics : ARRAY[0..2] OF LibSparkplug.SPB_FB_METRICS;   // Array with FBs
   for three metrics (0 .. 2) below the EoN
afbDevice : ARRAY[0..1] OF LibSparkplug.SPB_FB_DEVICE;      // Array with FBs
   for two devices (0 .. 1) below the EoN
abStorageBuffer : ARRAY[0..4999] OF USINT;            // Array with the storage
   buffer with 5000 bytes
stMqttParameter : LibSparkplug.SPB_MQTT_PARAMETERS;       // Structure with
   parameters for connection with the MQTT broker
xError : BOOL;                            // Output for indicating presence of an
   error
xIsConnected : BOOL;                        // Output for indicating connection to
    the MQTT borker
xIsBuffering : BOOL;                        // Output for indicating buffering
   mode
stStatus : LibSparkplug.SPB_STATUS;                // Structure with diagnostics
   about Sparkplug communication
eError : LibSparkplug.SPB_ERROR_CODE;              // Enumeration for indicating
   an error code detected by this FB

// Variables for instance of Device 0 below the EoN. Other inputs of instance
   passed directly in the code section.
xEnableDev0 : BOOL := TRUE;                    // Variable for enabling the FB
   for Device 0
afbDev0Metrics : ARRAY[0..1] OF LibSparkplug.SPB_FB_METRICS;  // Array with FBs
   for the two metrics (0 .. 1) below Device 0
eStatusDev0 : LibSparkplug.SPB_STATUS_DEVICE;          // Status for Device 0
eProtobufStatusDev0 : LibSparkplug.SPB_PROTOBUF_ERROR_CODE;   // Protobuf error
   codes for Device 0

// Variables for instance of Device 1 below the EoN. Other inputs of instance
   passed directly in the code section.
xEnableDev1 : BOOL := TRUE;                    // Variable for enabling the FB
   for Device 1
afbDev1Metrics : ARRAY[0..0] OF LibSparkplug.SPB_FB_METRICS;  // Array with FB
   for the one metric (0 .. 0) below Device 1
eStatusDev1 : LibSparkplug.SPB_STATUS_DEVICE;          // Status for Device 1
eProtobufStatusDev1 : LibSparkplug.SPB_PROTOBUF_ERROR_CODE;   // Protobuf error
   codes for Device 1

// Variables for Metric 0 directly below the EoN
rVar_Metric0_EoN : REAL;         // Variable reported by this metric
rVar_Metric0_EoN_aux : REAL;       // Auxiliary variable for reporting this
   metric by exception or deadband

// Variables for Metric 1 directly below the EoN
```

```
    iVar_Metric1_EoN : INT;          // Variable reported by this metric
    iVar_Metric1_EoN_aux : INT;        // Auxiliary variable for reporting this
        metric by exception or deadband

    // Variables for Metric 2 directly below the EoN
    xVar_Metric2_EoN : BOOL;         // Variable reported by this metric
    xVar_Metric2_EoN_aux : BOOL;       // Auxiliary variable for reporting this
        metric by exception or deadband

    // Variables for Metric 0 of Device 0
    lrVar_Metric0_Dev0 : LREAL;        // Variable reported by this metric
    lrVar_Metric0_Dev0_aux : LREAL;      // Auxiliary variable for reporting this
        metric by exception or deadband

    // Variables for Metric 1 of Device 0
    uiVar_Metric1_Dev0 : UINT;         // Variable reported by this metric
    uiVar_Metric1_Dev0_aux : UINT;       // Auxiliary variable for reporting this
        metric by exception or deadband

    // Variables for Metric 0 of Device 1
    liVar_Metric0_Dev1 : LINT;         // Variable reported by this metric
    liVar_Metric0_Dev1_aux : LINT;       // Auxiliary variable for reporting this
        metric by exception or deadband
    END_VAR
```

3.4.2. Code Section of POU EoN

```
// Configure MQTT parameters
stMqttParameter.sClientID := 'EON CONTROL';
stMqttParameter.sHostName := '192.168.201.140';
stMqttParameter.sUser := '';
stMqttParameter.sPass := '';
stMqttParameter.uiPort := 1883;
stMqttParameter.uiKeepAlive := 60;
stMqttParameter.xEnableTLS := FALSE;
stMqttParameter.sCertFileName := '';
stMqttParameter.sClientCert := '';
stMqttParameter.sClientKey := '';

// Configure variables for metric 0 of EoN
afbEonMetrics[0].in_sName := 'Metric 0 of EoN';
afbEonMetrics[0].in_pValue := ADR(rVar_Metric0_EoN);
afbEonMetrics[0].in_usiMaxPayloadSize := UINT_TO_USINT(SIZEOF(rVar_Metric0_EoN));
afbEonMetrics[0].in_eDataType := LibSparkplug.SPB_PROTOBUF_DATA_TYPES.Float;
afbEonMetrics[0].in_xEnableHist := TRUE;
afbEonMetrics[0].in_stPublishParameters.ePubMode := LibSparkplug.SPB_PUBLISH_MODE.RBE;
afbEonMetrics[0].in_stPublishParameters.pBufferVariable := ADR(rVar_Metric0_EoN_aux);
afbEonMetrics[0].in_stPublishParameters.rDeadBandValue := 0; // not used with RBE
afbEonMetrics[0].in_stPublishParameters.usiTrunc := 2;
afbEonMetrics[0].in_xTrigger := FALSE;

// Configure variables for metric 1 of EoN
afbEonMetrics[1].in_sName := 'Metric 1 of EoN';
afbEonMetrics[1].in_pValue := ADR(iVar_Metric1_EoN);
afbEonMetrics[1].in_usiMaxPayloadSize := UINT_TO_USINT(SIZEOF(iVar_Metric1_EoN));
afbEonMetrics[1].in_eDataType := LibSparkplug.SPB_PROTOBUF_DATA_TYPES.Int16;
afbEonMetrics[1].in_xEnableHist := TRUE;
afbEonMetrics[1].in_stPublishParameters.ePubMode := LibSparkplug.SPB_PUBLISH_MODE.DEAD_BAND_ABSOLUTE;
afbEonMetrics[1].in_stPublishParameters.pBufferVariable := ADR(iVar_Metric1_EoN_aux);
afbEonMetrics[1].in_stPublishParameters.rDeadBandValue := 3;
afbEonMetrics[1].in_stPublishParameters.usiTrunc := 0; // not used with integer variables
afbEonMetrics[1].in_xTrigger := FALSE;

// Configure variables for metric 2 of EoN
afbEonMetrics[2].in_sName := 'Metric 2 of EoN';
afbEonMetrics[2].in_pValue := ADR(xVar_Metric2_EoN);
afbEonMetrics[2].in_usiMaxPayloadSize := UINT_TO_USINT(SIZEOF(xVar_Metric2_EoN));
afbEonMetrics[2].in_eDataType := LibSparkplug.SPB_PROTOBUF_DATA_TYPES.Boolean;
afbEonMetrics[2].in_xEnableHist := TRUE;
afbEonMetrics[2].in_stPublishParameters.ePubMode := LibSparkplug.SPB_PUBLISH_MODE.RBE;
afbEonMetrics[2].in_stPublishParameters.pBufferVariable := ADR(xVar_Metric2_EoN_aux);
afbEonMetrics[2].in_stPublishParameters.rDeadBandValue := 0; // not used with boolean variables
afbEonMetrics[2].in_stPublishParameters.usiTrunc := 0; // not used with boolean variables
afbEonMetrics[2].in_xTrigger := FALSE;

```

Figure 8: Code of POU EoN - Part 1

```
// Configure variables for metric 0 of Device 0
afbDev0Metrics[0].in_sName := 'Metric 0 of Device 0';
afbDev0Metrics[0].in_pValue := ADR(lrVar_Metric0_Dev0);
afbDev0Metrics[0].in_usiMaxPayloadSize := UINT_TO_USINT(SIZEOF(lrVar_Metric0_Dev0));
afbDev0Metrics[0].in_eDataType := LibSparkplug.SPB_PROTOBUF_DATA_TYPES.Double;
afbDev0Metrics[0].in_xEnableHist := TRUE;
afbDev0Metrics[0].in_stPublishParameters.ePubMode := LibSparkplug.SPB_PUBLISH_MODE.RBE;
afbDev0Metrics[0].in_stPublishParameters.pBufferVariable := ADR(lrVar_Metric0_Dev0_aux);
afbDev0Metrics[0].in_stPublishParameters.rDeadBandValue := 0; // not used with RBE
afbDev0Metrics[0].in_stPublishParameters.usiTrunc := 2;
afbDev0Metrics[0].in_xTrigger := FALSE;

// Configure variables for metric 1 of Device 0
afbDev0Metrics[1].in_sName := 'Metric 1 of Device 0';
afbDev0Metrics[1].in_pValue := ADR(uiVar_Metric1_Dev0);
afbDev0Metrics[1].in_usiMaxPayloadSize := UINT_TO_USINT(SIZEOF(uiVar_Metric1_Dev0));
afbDev0Metrics[1].in_eDataType := LibSparkplug.SPB_PROTOBUF_DATA_TYPES.UInt16;
afbDev0Metrics[1].in_xEnableHist := TRUE;
afbDev0Metrics[1].in_stPublishParameters.ePubMode := LibSparkplug.SPB_PUBLISH_MODE.RBE;
afbDev0Metrics[1].in_stPublishParameters.pBufferVariable := ADR(uiVar_Metric1_Dev0_aux);
afbDev0Metrics[1].in_stPublishParameters.rDeadBandValue := 0; // not used with RBE
afbDev0Metrics[1].in_stPublishParameters.usiTrunc := 0; // not used with integer variables
afbDev0Metrics[1].in_xTrigger := FALSE;

// Configure variables for metric 0 of Device 1
afbDev1Metrics[0].in_sName := 'Metric 0 of Device 1';
afbDev1Metrics[0].in_pValue := ADR(liVar_Metric0_Dev1);
afbDev1Metrics[0].in_usiMaxPayloadSize := UINT_TO_USINT(SIZEOF(liVar_Metric0_Dev1));
afbDev1Metrics[0].in_eDataType := LibSparkplug.SPB_PROTOBUF_DATA_TYPES.Int64;
afbDev1Metrics[0].in_xEnableHist := TRUE;
afbDev1Metrics[0].in_stPublishParameters.ePubMode := LibSparkplug.SPB_PUBLISH_MODE.RBE;
afbDev1Metrics[0].in_stPublishParameters.pBufferVariable := ADR(liVar_Metric0_Dev1_aux);
afbDev1Metrics[0].in_stPublishParameters.rDeadBandValue := 0; // not used with RBE
afbDev1Metrics[0].in_stPublishParameters.usiTrunc := 0; // not used with integer variables
afbDev1Metrics[0].in_xTrigger := FALSE;

// Execute FB for the Device 0
afbDevice[0](
    in_xEnable:= xEnableDev0,
    in_sDeviceId:= 'Device 0',
    in_usiMaxMetrics:= 2,
    in_pMetrics:= ADR(afbDev0Metrics[0]),
    out_eStatus=> eStatusDev0,
    out_eProtobufStatus=> eProtobufStatusDev0);
```

Figure 9: Code of POU EoN - Part 2

```
94  // Execute FB for the Device 1
95  afbDevice[1](
96      in_xEnable:= xEnableDev1,
97      in_sDeviceId:= 'Device 1',
98      in_usiMaxMetrics:= 1,
99      in_pMetrics:= ADR(afbDev1Metrics[0]),
100     out_eStatus=> eStatusDev1,
101     out_eProtobufStatus=> eProtobufStatusDev1);
102
103 // Execute FB for the EoN
104 fbEoN(
105     in_xEnable:= xEnableEon,                    // Enable the FB
106     in_sGroupId:= 'GROUP 11',                   // Group ID for the EoN
107     in_sEonId:= 'PLC XP340',                    // EoN ID
108     in_sPrimaryHostId:= 'SCADA_Ignition_Host_1', // Primary host ID
109     in_pEonMetrics:= ADR(afbEonMetrics[0]),     // Pointer to the array with configuration of the metrics of the EoN
110     in_usiQtyEonMetrics:= 3,                    // Quantity of metrics directly below the EoN
111     in_pDevices:= ADR(afbDevice[0]),            // Pointer to the array with configuration of the devices of the EoN
112     in_usiQtyDevices:= 2,                       // Quantity of devices of the EoN
113     in_pStorageBuffer:= ADR(abStorageBuffer[0]), // Pointer to the array with the storage buffer
114     in_uiSizeBuffer:= UDINT_TO_UINT(SIZEOF(abStorageBuffer)),   // Size of the array with the storage buffer
115     in_stMqttParameters:= stMqttParameter,      // MQTT parameters
116     in_timTimePeriodic:= T#1S,                  // Period for detecting changes using publish modes RBE or DEAD_BAND_ABSOLUTE
117     out_xError=> xError,                        // Output for error presence indication
118     out_xIsConnected=> xIsConnected,            // Output for indicating connection with MQTT broker
119     out_xIsBuffering=> xIsBuffering,            // Output for indicating buffering mode
120     out_stStatus=> stStatus,                    // Output for indicating status (diagnostics)
121     out_eError=> eError);                       // Output for indicating an error code
122
123 // Manage the inputs in_xTrigger for all metrics
124 // Do not need to include those metrics where trigger publish mode will never be employed
125 afbEonMetrics[0]();
126 afbEonMetrics[0].in_xTrigger := FALSE;
127 afbEonMetrics[1]();
128 afbEonMetrics[1].in_xTrigger := FALSE;
129 afbEonMetrics[2]();
130 afbEonMetrics[2].in_xTrigger := FALSE;
131 afbDev0Metrics[0]();
132 afbDev0Metrics[0].in_xTrigger := FALSE;
133 afbDev0Metrics[1]();
134 afbDev0Metrics[1].in_xTrigger := FALSE;
135 afbDev1Metrics[0]();
136 afbDev1Metrics[0].in_xTrigger := FALSE;
```

Figure 10: Code of POU EoN - Part 3

The code between lines 1 and 83 is intended for configurations. One may think that this code could be executed only once during initialization. Nevertheless, executing this code in all cycles is recommended because of possible online changes.

The code between lines 123 and 136 is optional. Include code only for those metrics you want to report by setting the in_xTrigger input (two lines of code for each metric).

The following text box is a copy of the code section shown in the previous figure. It is intended to copy and paste this example, should you want to test it.

```
// Configure MQTT parameters
stMqttParameter.sClientID := 'EON CONTROL';
stMqttParameter.sHostName := '192.168.201.140';
stMqttParameter.sUser := '';
stMqttParameter.sPass := '';
stMqttParameter.uiPort := 1883;
stMqttParameter.uiKeepAlive := 60;
stMqttParameter.xEnableTLS := FALSE;
stMqttParameter.sCertFileName := '';
stMqttParameter.sClientCert := '';
stMqttParameter.sClientKey := '';

// Configure variables for metric 0 of EoN
afbEonMetrics[0].in_sName := 'Metric 0 of EoN';
afbEonMetrics[0].in_pValue := ADR(rVar_Metric0_EoN);
afbEonMetrics[0].in_usiMaxPayloadSize := UINT_TO_USINT(SIZEOF(rVar_Metric0_EoN))
    ;
```

```
afbEonMetrics[0].in_eDataType := LibSparkplug.SPB_PROTOBUF_DATA_TYPES.Float;
afbEonMetrics[0].in_xEnableHist := TRUE;
afbEonMetrics[0].in_stPublishParameters.ePubMode := LibSparkplug.
    SPB_PUBLISH_MODE.RBE;
afbEonMetrics[0].in_stPublishParameters.pBufferVariable := ADR(
    rVar_Metric0_EoN_aux);
afbEonMetrics[0].in_stPublishParameters.rDeadBandValue := 0; // not used with
    RBE
afbEonMetrics[0].in_stPublishParameters.usiTrunc := 2;
afbEonMetrics[0].in_xTrigger := FALSE;

// Configure variables for metric 1 of EoN
afbEonMetrics[1].in_sName := 'Metric 1 of EoN';
afbEonMetrics[1].in_pValue := ADR(iVar_Metric1_EoN);
afbEonMetrics[1].in_usiMaxPayloadSize := UINT_TO_USINT(SIZEOF(iVar_Metric1_EoN))
    ;
afbEonMetrics[1].in_eDataType := LibSparkplug.SPB_PROTOBUF_DATA_TYPES.Int16;
afbEonMetrics[1].in_xEnableHist := TRUE;
afbEonMetrics[1].in_stPublishParameters.ePubMode := LibSparkplug.
    SPB_PUBLISH_MODE.DEAD_BAND_ABSOLUTE;
afbEonMetrics[1].in_stPublishParameters.pBufferVariable := ADR(
    iVar_Metric1_EoN_aux);
afbEonMetrics[1].in_stPublishParameters.rDeadBandValue := 3;
afbEonMetrics[1].in_stPublishParameters.usiTrunc := 0; // not used with integer
    variables
afbEonMetrics[1].in_xTrigger := FALSE;

// Configure variables for metric 2 of EoN
afbEonMetrics[2].in_sName := 'Metric 2 of EoN';
afbEonMetrics[2].in_pValue := ADR(xVar_Metric2_EoN);
afbEonMetrics[2].in_usiMaxPayloadSize := UINT_TO_USINT(SIZEOF(xVar_Metric2_EoN))
    ;
afbEonMetrics[2].in_eDataType := LibSparkplug.SPB_PROTOBUF_DATA_TYPES.Boolean;
afbEonMetrics[2].in_xEnableHist := TRUE;
afbEonMetrics[2].in_stPublishParameters.ePubMode := LibSparkplug.
    SPB_PUBLISH_MODE.RBE;
afbEonMetrics[2].in_stPublishParameters.pBufferVariable := ADR(
    xVar_Metric2_EoN_aux);
afbEonMetrics[2].in_stPublishParameters.rDeadBandValue := 0; // not used with
    boolean variables
afbEonMetrics[2].in_stPublishParameters.usiTrunc := 0; // not used with boolean
    variables
afbEonMetrics[2].in_xTrigger := FALSE;

// Configure variables for metric 0 of Device 0
afbDev0Metrics[0].in_sName := 'Metric 0 of Device 0';
afbDev0Metrics[0].in_pValue := ADR(lrVar_Metric0_Dev0);
afbDev0Metrics[0].in_usiMaxPayloadSize := UINT_TO_USINT(SIZEOF(
    lrVar_Metric0_Dev0));
afbDev0Metrics[0].in_eDataType := LibSparkplug.SPB_PROTOBUF_DATA_TYPES.Double;
afbDev0Metrics[0].in_xEnableHist := TRUE;
afbDev0Metrics[0].in_stPublishParameters.ePubMode := LibSparkplug.
    SPB_PUBLISH_MODE.RBE;
afbDev0Metrics[0].in_stPublishParameters.pBufferVariable := ADR(
    lrVar_Metric0_Dev0_aux);
afbDev0Metrics[0].in_stPublishParameters.rDeadBandValue := 0; // not used with
```

```
    RBE
afbDev0Metrics[0].in_stPublishParameters.usiTrunc := 2;
afbDev0Metrics[0].in_xTrigger := FALSE;

// Configure variables for metric 1 of Device 0
afbDev0Metrics[1].in_sName := 'Metric 1 of Device 0';
afbDev0Metrics[1].in_pValue := ADR(uiVar_Metric1_Dev0);
afbDev0Metrics[1].in_usiMaxPayloadSize := UINT_TO_USINT(SIZEOF(
    uiVar_Metric1_Dev0));
afbDev0Metrics[1].in_eDataType := LibSparkplug.SPB_PROTOBUF_DATA_TYPES.UInt16;
afbDev0Metrics[1].in_xEnableHist := TRUE;
afbDev0Metrics[1].in_stPublishParameters.ePubMode := LibSparkplug.
    SPB_PUBLISH_MODE.RBE;
afbDev0Metrics[1].in_stPublishParameters.pBufferVariable := ADR(
    uiVar_Metric1_Dev0_aux);
afbDev0Metrics[1].in_stPublishParameters.rDeadBandValue := 0; // not used with
    RBE
afbDev0Metrics[1].in_stPublishParameters.usiTrunc := 0; // not used with integer
     variables
afbDev0Metrics[1].in_xTrigger := FALSE;

// Configure variables for metric 0 of Device 1
afbDev1Metrics[0].in_sName := 'Metric 0 of Device 1';
afbDev1Metrics[0].in_pValue := ADR(liVar_Metric0_Dev1);
afbDev1Metrics[0].in_usiMaxPayloadSize := UINT_TO_USINT(SIZEOF(
    liVar_Metric0_Dev1));
afbDev1Metrics[0].in_eDataType := LibSparkplug.SPB_PROTOBUF_DATA_TYPES.Int64;
afbDev1Metrics[0].in_xEnableHist := TRUE;
afbDev1Metrics[0].in_stPublishParameters.ePubMode := LibSparkplug.
    SPB_PUBLISH_MODE.RBE;
afbDev1Metrics[0].in_stPublishParameters.pBufferVariable := ADR(
    liVar_Metric0_Dev1_aux);
afbDev1Metrics[0].in_stPublishParameters.rDeadBandValue := 0; // not used with
    RBE
afbDev1Metrics[0].in_stPublishParameters.usiTrunc := 0; // not used with integer
     variables
afbDev1Metrics[0].in_xTrigger := FALSE;

// Execute FB for the Device 0
afbDevice[0](
  in_xEnable:= xEnableDev0,
  in_sDeviceId:= 'Device 0',
  in_usiMaxMetrics:= 2,
  in_pMetrics:= ADR(afbDev0Metrics[0]),
  out_eStatus=> eStatusDev0,
  out_eProtobufStatus=> eProtobufStatusDev0);

// Execute FB for the Device 1
afbDevice[1](
  in_xEnable:= xEnableDev1,
  in_sDeviceId:= 'Device 1',
  in_usiMaxMetrics:= 1,
  in_pMetrics:= ADR(afbDev1Metrics[0]),
  out_eStatus=> eStatusDev1,
  out_eProtobufStatus=> eProtobufStatusDev1);
```

```
// Execute FB for the EoN
fbEoN(
  in_xEnable:= xEnableEon,              // Enable the FB
  in_sGroupId:= 'GROUP 11',             // Group ID for the EoN
  in_sEonId:= 'PLC XP340',              // EoN ID
  in_sPrimaryHostId:= 'SCADA_Ignition_Host_1',  // Primary host ID
  in_pEonMetrics:= ADR(afbEonMetrics[0]),     // Pointer to the array with
   configuration of the metrics of the EoN
  in_usiQtyEonMetrics:= 3,              // Quantity of metrics directly below the
   EoN
  in_pDevices:= ADR(afbDevice[0]),         // Pointer to the array with
   configuration of the devices of the EoN
  in_usiQtyDevices:= 2,                 // Quantity of devices of the EoN
  in_pStorageBuffer:= ADR(abStorageBuffer[0]),  // Pointer to the array with the
    storage buffer
  in_uiSizeBuffer:= UDINT_TO_UINT(SIZEOF(abStorageBuffer)), // Size of the array
    with the storage buffer
  in_stMqttParameters:= stMqttParameter,       // MQTT parameters
  in_timTimePeriodic:= T#1S,             // Period for detecting changes using
   publish modes RBE or DEAD_BAND_ABSOLUTE
  out_xError=> xError,                 // Output for error presence indication
  out_xIsConnected=> xIsConnected,         // Output for indicating connection
   with MQTT broker
  out_xIsBuffering=> xIsBuffering,         // Output for indicating buffering
   mode
  out_stStatus=> stStatus,              // Output for indicating status (
   diagnostics)
  out_eError=> eError);                // Output for indicating an error code

// Manage the inputs in_xTrigger for all metrics
// Do not need to include those metrics where trigger publish mode will never be
    employed
afbEonMetrics[0]();
afbEonMetrics[0].in_xTrigger := FALSE;
afbEonMetrics[1]();
afbEonMetrics[1].in_xTrigger := FALSE;
afbEonMetrics[2]();
afbEonMetrics[2].in_xTrigger := FALSE;
afbDev0Metrics[0]();
afbDev0Metrics[0].in_xTrigger := FALSE;
afbDev0Metrics[1]();
afbDev0Metrics[1].in_xTrigger := FALSE;
afbDev1Metrics[0]();
afbDev1Metrics[0].in_xTrigger := FALSE;
```

3.5. Example of Authentication Configuration

Authentication means to use a user name and a password for authorizing the connection between the EoN to the MQTT broker.

This section gives an example of authentication configuration using the MQTT broker Mosquitto version 2.0.18.

In addition, it is necessary to configure a user name and a password in the host applications that connect to this Mosquitto MQTT broker. This is not covered in this example. Please read the user manual of the host application for this purpose.

Inittially define a user name and a password that will be passed in the EoN application, using the fields sUser and sPass of input in_stMqttParameters of Function Block SPB_FB_SPARKPLUG. For instance:

- sUser = 'xp340'
- sPass = 'altus'

The remaining steps define how to configure the authentication in MQTT broker Mosquitto version 2.0.18:

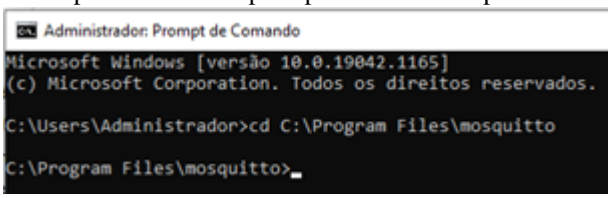
1. Create a text file named "passwd.txt" in the same path where Mosquitto is installed (for instance: `C:\ProgramFiles\mosquitto`). This file must contain at least the following line:

```
xp340:altus
```

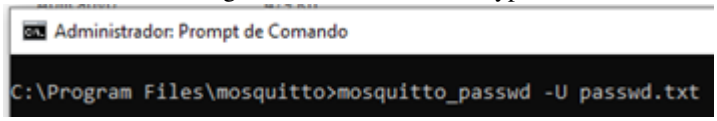
2. Additional lines can be inserted in the file "passwd.txt", for creating more user names and passwords that allow connection. So, different clients can connect using different passwords. For instance:

```
xp340:altus
host_1:scada
```

3. Now open a command prompt and select the path where Mosquito installed:



4. Execute the following command that will encrypt the file with user names and passwords:

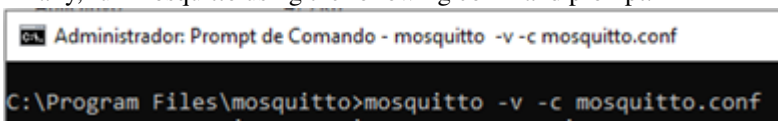


5. If you open the file passwd.txt again, you will see something like this:

6. Now it is necessary to make some changes in the configuration file (for instance, mosquitto.conf). The two following lines must be adjusted in this file for requesting authentication and defining the file where the user names and passwords are defined:

```
allow_anonymous false
password_file C:\Program Files\mosquitto\passwd.txt
```

7. Finally, run mosquitto using the following command prompt:



3.6. Example of TLS Security Configuration

TLS security means to use an encrypted communication between EoN and the MQTT broker (TLS encryption versions 1.0, 1.1 or 1.2).

This section gives an example of TLS version 1.2 configuration using the MQTT broker Mosquitto version 2.0.18.

In addition, it is necessary to configure TLS security in the host applications that connect to this Mosquitto MQTT broker. This is not covered in this example. Please read the user manual of the host application for this purpose.

The following subsections define the steps for this configuration.

3.6.1. Adjust Clock and Keep them Synchronized

Adjust the clock of computers and PLCs involved in the process before starting the configuration. This is necessary because certificate files expire, and the expiration date is calculated relative to the current date and time.

Afterwards, keep the EoN and host applications with correct date and time for avoiding unexpected expiration of the certificate files. For instance, use the STNP synchronization capability of computers and EoN.

3.6.2. Generate Certificates

Certificate files must be generated for using TLS security. These files must be installed in the MQTT server (MQTT broker) and in the MQTT clients (EoN and host applications).

The following certificate files must be generated:

- **ca.crt:** this is the root certificate file, that must be installed in the MQTT server (MQTT broker) and in all the MQTT clients (EoN and host applications).
- **server.crt:** this is a specific certificate file for the server, that must be installed only in the MQTT server (MQTT broker).
- **server.key:** this is a specific key file for the server, that must be installed only in the MQTT server (MQTT broker).
- **eon.crt:** this is a specific certificate file for a MQTT client (in this case the Nexto PLC as an EoN). It must be installed only in the MQTT client (Nexto PLC as an EoN).
- **eon.key:** this is a specific key file for a MQTT client (in this case the Nexto PLC as an EoN). It must be installed only in the MQTT client (Nexto PLC as an EoN).

For other MQTT clients like host applications, two specific files (.crt and .key) must be generated. The TLS configuration for host applications is not covered by this example (see user manual of the host application).

For generating the aforementioned certificate files, we used the following version of software OpenSSL:

```
C:\Certificates>openssl version
OpenSSL 1.1.1w  11 Sep 2023

C:\Certificates>
```

After installing the software, add the path `C:\ProgramFiles\OpenSSL-Win64\bin` to the PATH environment variable of Windows, so that you can execute OpenSSL from any folder.

The following steps describe how to generate the 5 aforementioned certificate files in a folder called `C:\Certificates`. Note that some additional files are generated (a few intermediary files that will not be used).

Step 1: Open a command prompt and go to folder `C:\Certificates` where the certificates will be generated:

```
Prompt de Comando
Microsoft Windows [versão 10.0.19044.3086]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\brune>cd \Certificates

C:\Certificates>
```

Step 2: Generate intermediary file "ca.key":

```
C:\Certificates>openssl genrsa -out ca.key 4096
Generating RSA private key, 4096 bit long modulus (2 primes)
.....................................................
.....................................................
.....................++++
e is 65537 (0x010001)
```

Step 3: Generate file "ca.crt":

The parameter "-days" define the validity of the certificate in days (in this example 365 days). After executing the command, it asks for several information fields, like country name. It is not necessary to inform most of these fields, except the field "Common Name" (in this example: CA-Entity).

```
C:\Certificates>openssl req -new -x509 -days 365 -key ca.key -out ca.crt
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:CA-Entity
Email Address []:
```

Step 4: Generate file "server.key" for the MQTT broker:

```
C:\Certificates>openssl genrsa -out server.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.....................................................................
.....................................................................
e is 65537 (0x010001)
```

Step 5: Generate file "eon.key" for the EoN:

```
C:\Certificates>openssl genrsa -out eon.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.....................................+++++
.....................................................................
e is 65537 (0x010001)
```

Step 6: Generate intermediary file "server.csr" for the MQTT broker:

After executing the command, it asks for several information fields, like country name. It is not necessary to inform most of these fields, except the field "Common Name" that must inform the IP adress of the machine with the MQTT broker (in this example: 192.168.201.141).

```
C:\Certificates>openssl req -new -key server.key -out server.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:192.168.201.141
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Step 7: Generate intermediary file "eon.csr" for the EoN:

After executing the command, it asks for several information fields, like country name. It is not necessary to inform most of these fields, except the field "Common Name" that must inform the IP adress of the EoN (in this example: 192.168.201.50).

```
C:\Certificates>openssl req -new -key eon.key -out eon.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:192.168.201.50
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Step 8: Generate file "server.crt" for the MQTT broker:

```
C:\Certificates>openssl x509 -req -days 365 -in server.csr -CA ca.crt -CAkey ca.key -set_serial 01 -out server.crt
Signature ok
subject=C = AU, ST = Some-State, O = Internet Widgits Pty Ltd, CN = 192.168.201.141
Getting CA Private Key
```

Step 9: Generate file "eon.crt" for the EoN:

```
C:\Certificates>openssl x509 -req -days 365 -in eon.csr -CA ca.crt -CAkey ca.key -set_serial 01 -out eon.crt
Signature ok
subject=C = AU, ST = Some-State, O = Internet Widgits Pty Ltd, CN = 192.168.201.50
Getting CA Private Key
```

Step 10: Verify file "server.crt" for the MQTT broker:

```
C:\Certificates>openssl verify -purpose sslserver -CAfile ca.crt server.crt
server.crt: OK
```

Step 11: Verify file "eon.crt" for the EoN:

```
C:\Certificates>openssl verify -purpose sslserver -CAfile ca.crt eon.crt
eon.crt: OK
```

Step 12: Check the files in the folder:

At the end, the following files must be in the folder. The .csr files, and ca.key, are intermediary files that will not be used in the following steps.

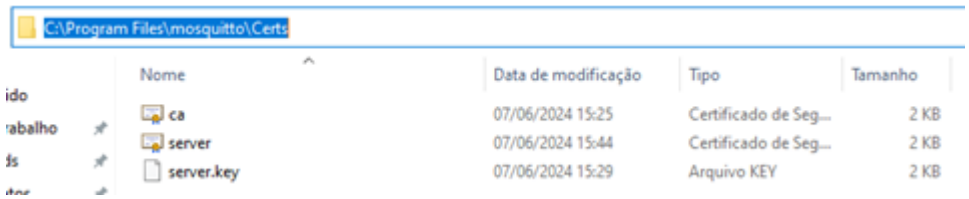
```
C:\Certificates>dir
 O volume na unidade C não tem nome.
 O Número de Série do Volume é 9AC3-FFB9

 Pasta de C:\Certificates

07/06/2024  15:46    <DIR>          .
07/06/2024  15:46    <DIR>          ..
07/06/2024  15:25             2.024 ca.crt
07/06/2024  15:17             3.294 ca.key
07/06/2024  15:46             1.530 eon.crt
07/06/2024  15:42             1.010 eon.csr
07/06/2024  15:31             1.706 eon.key
07/06/2024  15:44             1.534 server.crt
07/06/2024  15:36             1.010 server.csr
07/06/2024  15:29             1.702 server.key
               8 arquivo(s)         13.810 bytes
               2 pasta(s)   319.059.087.360 bytes disponíveis
```

3.6.3. Install Certificate Files in the MQTT Broker

Select the folder where the MQTT Broker Mosquitto is installed (e.g.: `C:\ProgramFiles\mosquitto`), and create the folder "Certs" (if not created yet).

After this, copy the three following files to the folder `C:\ProgramFiles\mosquitto\Certs`:

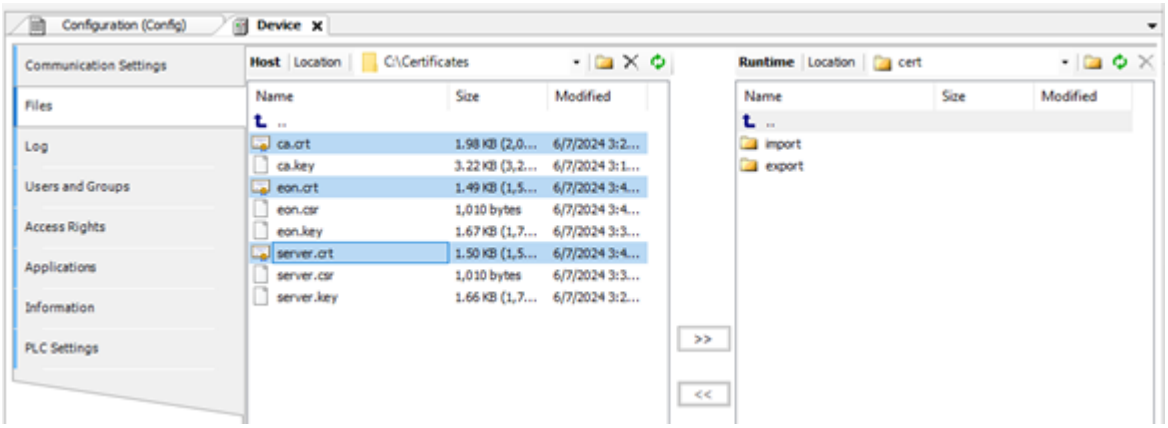
- ca.crt
- server.crt
- server.key



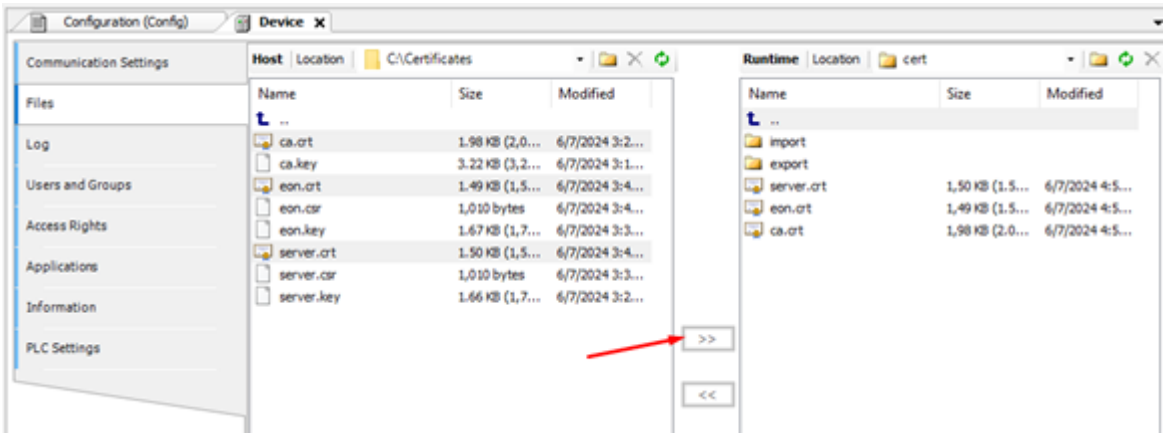
3.6.4. Install Certificate Files in the EoN

Using the Mastertool programming system, select *Device/Files* and select the following files from `C:\Certificates` in the left panel, and select the folder "cert" in the right panel:

- ca.crt
- eon.crt
- eon.key



After this, copy these three files to the folder "cert" of the EoN.



3.6.5. Edit the Configuration File of MQTT Broker Mosquitto

For using TLS security, some changes must be made in the configuration file of Mosquitto (`C:\ProgramFiles\mosquitto\mosquitto.conf`).

1. Change the listener port to 8883 (the normal value without TLS is 1883):

```
listener 8883
```

2. Define the name and location of the certificate files:

```
cafile C:\Program Files\mosquitto\Certs\ca.crt
certfile C:\Program Files\mosquitto\Certs\server.crt
keyfile C:\Program Files\mosquitto\Certs\server.key
```

3. Request to use certificate files:

```
require_certificate true
```

3.6.6. Adjust the EoN Application

For using the TLS configuration of this example, the following fields of input in_stMqttParameters must be adjusted:

- uiPort = 8883
- xEnableTLS = TRUE
- sCertFileName = 'ca.crt'
- sClientCert = 'eon.crt'
- sClientKey = 'eon.key'

3.7. Effects of Online Change

Sometimes the user may download a new application using an online change. The changes in this new application may or may not be related to the Sparkplug configuration.

To avoid problems like exceptions, the Function Block SPB_FB_SPARKPLUG detects an online change and then commands an automatic reconfiguration. This reconfiguration behaves very similar to a power-on configuration, or to a reconfiguration commanded by the user creating a rising edge in the input in_xEnable of the Function Block SPB_FB_SPARKPLUG.

However, an automatic reconfiguration after an online change tries to preserve the historical data in the storage buffer (see section Buffering Capability of Nexto Controllers). The storage buffer will only be preserved if the following conditions are true:

- The address of the storage buffer has not changed (input in_pStorageBuffer of Function Block SPB_FB_SPARKPLUG).
- The size of the storage buffer has not changed (input in_uiSizeBuffer of Function Block SPB_FB_SPARKPLUG).