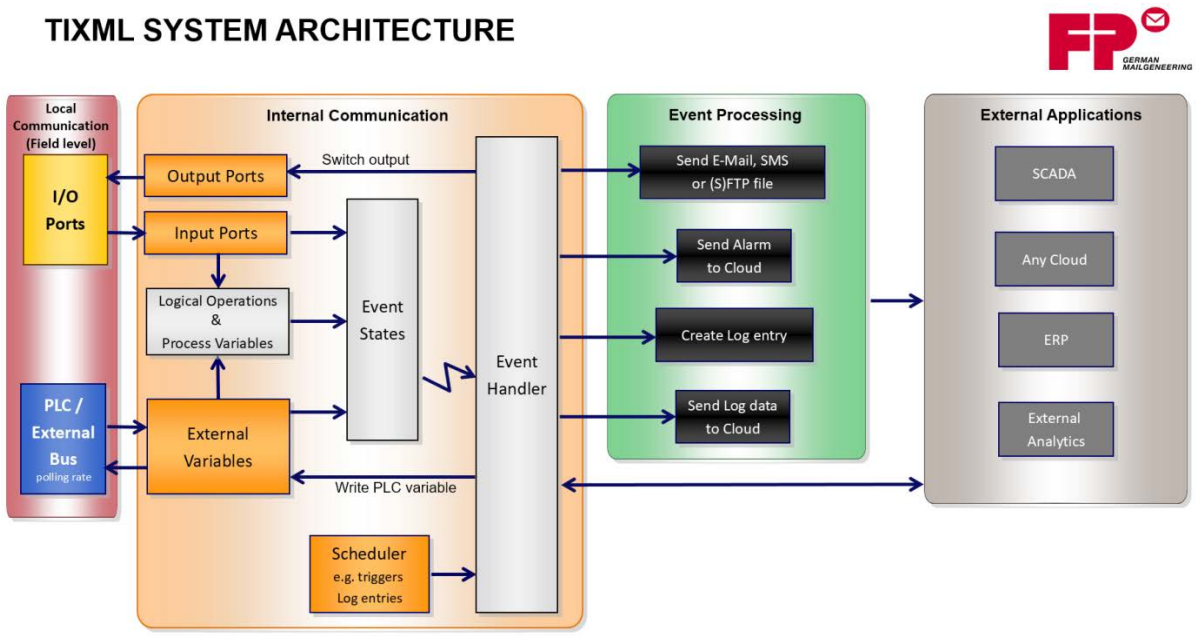
FP PLC TiXML Manual



For Linux series: OTGuard Hx600
ENGuard Hx600
ENGuard Wx600
ENGuard Wx550

For Legacy series: Hx100/400

Version: 3.7.1

www.inovolabs.com

Publication date: 23/02/2021

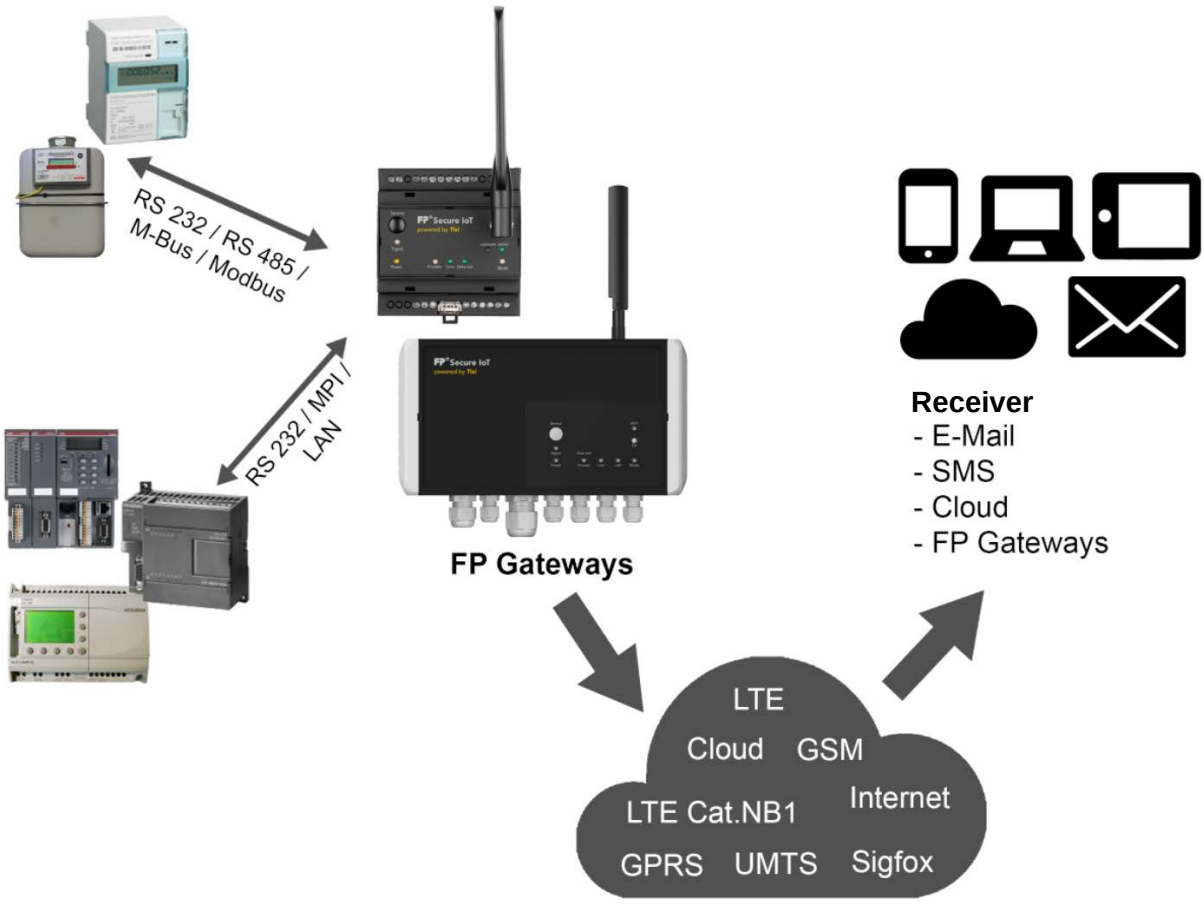
Table of contents

1 Overview

FP IoT gateways can be integrated into existing systems with minimum effort (“retrofit”).

The communication protocols for widely-used PLC systems already exist in FP IoT gateways and therefore, the PLC does not require adjustment. Other PLC systems can control the FP gateway via simple TiXML text commands.

Example of an application for FP gateways:



This manual describes the settings required for connection to a PLC. In addition, it provides an overview of the PLC systems that are currently supported and their scope of variables, as well as commands for formatting PLC variable values.

Most of the protocols and controllers described here apply to FP IoT gateways as of the sixth generation, including the wall box. The SML protocol is only available on eighth generation FP IoT gateways.

2 Setting parameters for FP IoT gateways for PLC systems

2.1 Configuring PLC and bus systems

In order to enable FP IoT gateways to communicate with the PLC, parameters must be set for the FP IoT gateway's external database. This description requires basic knowledge of TiXML language and the FP IoT gateway.

The PLC configuration database is `PROCCFG/External` and looks like this:

```
<External>
   <Bus _="Bus" BusId="BusId" {Name="Alias"} protocol="Proto"
    type="BType" {baud="Speed"} {format="Dataformat"}
    {handshake="handshake"} {Mem="Memory"} [TS="OwnID"]
    MAXADR="Range"] [GUF="Factor"] [RC="Retries"]>
    {<Condition _="Name" Variable="Path" Pollrate="Rate"/>}
       <Device _="ID" {Name="Alias"} {Pollrate="Rate TUnit"}
        [CharTimeout="CT TUnit "] [Pause="Wait TUnit "]
        [Timeout="Timeout TUnit"] [DWordInc="AI"] [DwordSwap="Swap"]
        [ForceSingleWordWrite="Funct"] [PrimaryAddr="PA"]
        [SecondaryAddr="SA"] [FabricationAddr="FA"]
        [ManufactoryCode="MC"] [Generation="Gen"] [Medium="Med"]
        [devType="DType"] [UseCache="Cache"]
        [MaxElements="Elements"] {Condition="Name"}>
           <VName _="VType" {simpleType="BasicType" } [exp="Exp"]
           [precision="Precision"] [size="ArraySize" ] acc="Rights
           Storage" [ind="Index"] [subind="Index2"] [no="Array"]
           {def="default"} {multip="Factor"}  {format="Format"}
           {write="wFunct"} {read="rFunct"}/>
      </Device>
   </Bus>
</External>
```

Attributes in **{…}** are optional.
Attributes in **[…]** are only required for certain devices or bus systems.

Overview of possible parameters:

<Bus> parameters

Bus Defines the interface to which the PLC is connected. Possible values:

COM1	PLC to COM1 (only Hx devices as of FW 1.80.0.0, requires BusId)
COM2	PLC to COM2 (only Hx devices as of FW 1.80.0.0, requires BusId)
COM3	PLC to COM3 (normally M-bus devices; only certain devices)
COM4	PLC to COM4 (RS485 devices; only certain devices)
COM5	PLC to COM5 (RS485 devices; only certain devices)
ETH	PLC to LAN interface (Modbus TCP, Siemens S7)

BusId Enables the PLC bus to be addressed regardless of the interface.
Must not be confused with "Name".

Alias Enables the PLC bus to be named and therefore enables addressing regardless of the interface. Must not be confused with “BusId”. (max. 20 alpha-numeric characters, no special characters, must not begin with a number)

Proto Determines the PLC protocol or bus protocol.
Depends on the device, PLC or bus system connected.
The valid values are specified in section 3.

BType Determines the communication mode towards the PLC: `Master` or `slave`. The valid values are specified in section 3.

Speed Determines the baud rate in bps between the FP IoT gateway and the PLC or describes the baud rate on the bus system (e.g. `19200`). The valid values are specified in section 3.

Dataformat Determines the data formal on the serial interface. The data format is selected according to the PLC or bus system used (e.g. 8E1 for S7-200). The valid values are specified in section 3.

Syntax: DataBitsParityBitsStopBits

DataBits:

8..8 data bits
7..7 data bits

ParityBits:

N..no parity bit
E..even parity
O..odd parity

StopBits:

1..1 stop bit
2..2 stop bits

Handshake Determines the software or hardware handshake between the device and the PLC or bus system.

None	Communication without handshake
XONXOFF	Software handshake
XONXOFFPASS	Software handshake, XONXOFF is forwarded to the application
RTSCTS	Hardware handshake with RTS CTS
DTRDSR	Hardware handshake with DTR DSR
HALF	Half-duplex RS 485 communication
FULL	Full-duplex RS 485/422
HALFX	Half-duplex RS 485 communication with XON XOFF
FULLX	Full-duplex RS 485/422 with XON XOFF
noDTR	deactivates the DTR

Memory Defines how much memory (in bytes) is reserved for the bus. A total of up to 20,000,000 bytes (20 MB, only for Linux devices as of FW 5.1.6.8) are available for all possible bus definitions.

OwnID Defines the device's station number. Information on whether this attribute is used and which values are valid is provided in section 3.

Range Maximum number of stations to be queried. Information on whether this attribute is used and which values are valid is provided in section 3.

Factor "Gap update factor" to detect further slaves. Information on whether this attribute is used and which values are valid is provided in section 3.

Retries Number of repetitions in the event of communication errors. Information on whether this attribute is used and which values are valid is provided in section 3.

Up to 5 different PLC systems can be defined depending on the model. If an interface is used for MPI communication, parameters must first be set for this in the external.

<Device> parameters

ID Defines the station number for the device or PLC to be queried. This must be identical to the name set in the PLC station (device station) (e.g. 1). Some bus systems allow several stations (network).

Alias Enables the PLC station to be named and therefore enables addressing regardless of the station number.

Rate FP IoT gateway query cycle in master mode or communication timeout in slave mode in the specified time unit (see *TUnit*, the default is 60s).

CT Timeout between the characters in the specified time unit (see *TUnit)* Information on whether this attribute is used and which values are valid is provided in section 3.

Wait Pause between the notifications in the specified time unit (see *TUnit)* Information on whether this attribute is used and which values are valid is provided in section 3.

Timeout Timeout for the response in the specified time unit (see *TUnit)* Information on whether this attribute is used and which values are valid is provided in section 3.

AI Address increment between two subsequent DWORDSs (2, only Modbus RTU). Information on whether this attribute is used and which values are valid is provided in section 3.

Swap Must be set if low before high word is sent in DWORD (0, only Modbus RTU). Information on whether this attribute is used and which values are valid is provided in section 3.

Funct Set if Funct 0x06 instead of 0x10 is to be used for individual WORD writing (0, only Modbus RTU). Information on whether this attribute is used and which values are valid is provided in section 3.

PA Primary address (only M-bus). Information on whether this attribute is used and which values are valid is provided in section 3.

SA Secondary address (only M-bus). Information on whether this attribute is used and which values are valid is provided in section 3.

FA Fabrication address (only M-bus). Information on whether this attribute is used and which values are valid is provided in section 3.

MC Manufacturer code (only M-bus). Information on whether this attribute is used and which values are valid is provided in section 3.

Gen Device generation (only M-bus). Information on whether this attribute is used and which values are valid is provided in section 3.

Med Device medium (only M-bus). Information on whether this attribute is used and which values are valid is provided in section 3.

DType CPU type (only Mitsubishi FX). Information on whether this attribute is used and which values are valid is provided in section 3.

Cache If the value is set to 0, combining subsequent variables into block queries (caching) is deactivated. All variables are collected in individual queries. (Default: 1, only Modbus RTU). Information on whether this attribute is used and which values are valid is provided in section 3.

Elements Number of variables that are queried per Modbus telegram (caching). Information on whether this attribute is used and which values are valid is provided in section 3.

Name Name of the condition to change the poll rate.

TUnit Time unit for time data (ms, s, m, h).

<Condition> parameters

Conditions are used to change the poll rate depending on a condition

Name Name of the condition that is referred to in the device section.

Path Variable by which the condition is considered met (=1). Entered without reference character.

Rate FP IoT gateway query cycle in master mode or communication timeout in slave mode (e.g. `1s,2m,6h`)

<Variables> parameters

For performance reasons, we recommend defining fewer than 1000 variables. There must not be any duplicate variables with an identical type and index (e.g. two markers with 2 different format instructions) in a station (device).

VName Name of the variable for which parameters are set in the following. The name may be up to 30 characters in length.

VType Determines the variable type for the PLC or bus protocol, e.g. a meter or a marker. The valid types are specified in section 3.

BasicType Determines the basic type for the variable as used in the application. The basic type determines the corresponding formatting options and the native display in the TiXML protocol. Several basic types (“simpleType” attribute) can be assigned to a variable type (attribute = “_“); one of these must be selected for each variable entry. Further attributes must be specified for different basic types (see “exp”, “size”). The following values are possible:

BasicType value	Meaning	Further attrib.	Native display examples
Uint8	8-bit unsigned value (0-255)	exp	“123” (exp = 0) “12.3” (exp = -1) “12300” (exp = 2) “0.0123” (exp = -4)
Uint16	16-bit unsigned value (0-65535)	exp	“12345” (exp = 0) “1234.5” (exp = -1) “1234500” (exp = 2)
Uint32	32-bit unsigned value (0-4294967295)	exp	“1234567” (exp = 0) “123456.7” (exp = -1) “123456700” (exp = 2)
Int8	8-bit signed value (-128-+127)	exp	“123” (exp=0) “-123” (exp=0) “1.23” (exp=-2) “12300” (exp=2)
Int16	16-bit signed value (-32768-32767)	exp	“-12345” (exp = 0) “ 12345” (exp = 0) “-1234.5” (exp = -1) “ 1234.5” (exp = -1) “-1234500” (exp = 2) “ 1234500” (exp = 2)
Int32	32-bit signed value (-2147483648-2147483647)	exp	“-1234567” (exp = 0) “ 1234567” (exp = 0) “-123456.7” (exp = -1) “ 123456.7” (exp = -1) “-123456700” (exp = 2) “ 123456700” (exp = 2)
String	Text (0…size character)	size	"This is a text” (only the first 100 characters)
Blob	Binary data array (0...size byte) currently not supported	size	“AF037FFF” Byte-by-byte hexadecimal (only the first 100 bytes)
Bit	Digital value (0-1)		“0” “1”
Float	Floating-point number, single precision ($\pm 3.402823466 * 10^{38}$)	exp	12.34567 -0.001234 -0.123456 E-7 0.123456 E+7 (Exponential representation when |Exponent| > 6)
Double	Floating-point number, double precision ($\pm 1.7976931348623158 * 10^{308}$)	exp	12.345678910 -0.0012345678 -0.1234567890 E-11 0.1234567890 E+11 (Exponential representation when |Exponent| > 11)

Rights Defines the FP IoT gateway's access rights to the variable:

R Read access
W Write access
RW Read/write access

The valid values for this attribute value depend on the variable type and are specified in section 3.

Storage Defines further memory and access options:

L (RWL) joint memory (only for S-bus variables with read/write access)

A (RA, WA, RWA) Activates active variable access (Siemens)

C Activates cached access to variable blocks if the general cache was deactivated via "UseCache" (see above). If the indicator is set, the applicable variable is not queried immediately , but checked whether the following can also still be read along with it in the query. If this has a different type, the query is still performed individually.

Further access options can be defined for different PLCs or devices. The valid values for this attribute value depend on the variable type and are specified in section 3.

Index Variable address. This depends on the selected variable type and must match the parameters set in the device or the bus. The valid addresses are specified in section 3.

Index2 The variable bus address depends on the selected variable type and must match the parameters set in the device or the bus. The valid addresses are specified in section 3.

Array Number of elements that are queried as an array.

Default Start value for the variable. For variables with write access, this is written during each (!) system start to the PLC. For variables with read access, the value is used when starting the system until the device receives the actual value from the PLC. The start value must be specified so that it is suitable for "exp" and simpleType (see the table in section 2.3).

ArraySize 1. simpleType = String *(see BasicType)*
The maximum number of ASCII characters in a text value (only valid for the simpleType="String" type, 0-65535). In null-terminated strings, the null character must be included in the calculation. The value depends on the device type or the bus protocol (optional, depends on the basic type, device or bus system).

2. simpleType= Blob (see *BasicType*)
The maximum number of bytes in a byte array (only valid for the simpleType="Blob" type, 0-65535). The value depends on the device type or the bus protocol (optional, depends on the basic type, device or bus system).

Factor The value received from the device is multiplied by this factor before it is processed further. An optional offset can also be specified:

valueGateway = Factor * valueDevice
valueDevice = 1/Factor * valueGateway + Offset

The factor is indicated by a break, e.g.: "1/1000+20" or "3600/1-10". The denominator and the meter are not permitted to be zero.
Use of this attribute depends on the variable type.
The valid values are specified in section 3.

Example

multip="10/32-20" multip="40/100" multip="1/300+50"

Exp Exponent to basis 10 that describes the resolution of a fixed-point number with the **simpleType = Uint8, Uint16, Uint32, Int8, Int16, Int32 type (see *BasicType*)**.
The values saved in the FP IoT gateway are multiplied by 10 exp(*Exp*) (after the *Factor* was applied) in order to determine the parameter's value.

valueParameter = 10^{Exp} * valueGateway.

The exponent therefore determines the position of the decimal point for a fixed-point number.

The following values are available for the exponent

Exp value	Description
-6	Resolution = 0.000001
-5	Resolution = 0.00001
-4	Resolution = 0.0001
-3	Resolution = 0.001
-2	Resolution = 0.01
-1	Resolution = 0.1
0	Resolution = 1 (default)
1	Resolution = 10
2	Resolution = 100
3	Resolution = 1000
4	Resolution = 10000
5	Resolution = 100000
6	Resolution = 1000000

Precision Precision of the integer representation for a value with **simpleType = Float , Double (see BasicType)**.

The value saved in the FP IoT gateway is multiplied by 10 exp(*Exp*) in order to convert the value to integer representation. Integer representation is used when calculating the process variables, e.g using the commands (GT, LT etc.). It therefore specifies the precision when calculating the process variables and must be defined depending on the application.

Integer representation = whole number(10^{Exp} * valueParameter).

The values correspond to the following table.

Precision Value	Description
-6	Factor = 0.000001
-5	Factor = 0.00001
-4	Factor = 0.0001
-3	Factor = 0.001
-2	Factor = 0.01
-1	Factor = 0.1
0	Factor = 1 (default)
1	Factor = 10

Precision Value	Description
2	Factor = 100
3	Factor = 1000
4	Factor = 10000
5	Factor = 100000
6	Factor = 1000000

Format The format options designate the default formatting for the value as it is output, e.g. in e-mails or in the Get command (see section 5).

wFunct Function code that is used when writing a variable. Information on whether this attribute is used and which values are valid is provided in section 3.

rFunct Function code that is used when reading a variable. Information on whether this attribute is used and which values are valid is provided in section 3.

2.2 Reading device variables

As with all other variable values, the value for a variable from the external database can be queried using the TiXML "Get" command. The "format" attribute is added to this command for the variables from the external database.

Command:

```
[<Get _="Vpath" format ="Format" ViewProperties="ViewProperties"/>]
```

Overview of possible parameters (values written in italics):

Vpath Path to address the parameter, the device or the bus.

Format Format specification, the following values are possible:

Format value	Description
„integer“	1. simpleType = Uint8, Uint16, Uint32, Int8, Int16, Int32 (see 2.1 *BasicType*). The value is output in the integer representation. The integer representation is calculated using the exponent defined for the variable from the variable's value (see section 2.1 *Exp*): Value in integer representation = 10^{-Exp} * value 2. simpleType = float, double (see 2.1 *BasicType*). The value is output in the integer representation. The integer representation is calculated using the precision defined for the variable from the variable's value (see section 2.1. *Precision*): Value in integer representation = $10^{-Precision}$ * value 3. All other data types (see 2.1 *BasicType*). The value is output in the native representation (native representation see section 2.1 *BasicType*).

Format value	Description
Empty("") or "native"	The value is output in the native representation (native representation see section 2.1 *BasicType*)
Format string	The value is output according to the specified formatting (see section 5, Formatting PLC variable values).
ViewProperties	If the AddProperties="Name,TimeStamp" attribute is defined in the bus definition, a Get command can use ViewProperties to output the plain text name for the bus (if defined) and a time stamp. The time stamp provides the time of the last successful variable reading.

If the format attribute is missing, the value is output in the format that was specified as the default format in the variable definition (see section 2.1). If no default format has been defined, the value is output in the native representation in this case (native representation see section 2.1 *BasicType*).

Response:
`[<Get _="Value" />]`

Value Value in the set formatting or representation.

2.3 Setting device variables

As with all other variable values, the value for a variable from the `external` database can be queried using the TiXML "Set" command. The "**format**" attribute is added to this command for the variables from the `external` database.

Command:
`[<Set _="Vpath" value="Value" format ="Format" />]`

Overview of possible parameters (values written in italics):

Vpath Path to address the parameter.
Value Parameter value.
Format Format specification, the following values are possible:

Format value	Description
„integer"	1. simpleType = Uint8, Uint16, Uint32, Int8, Int16, Int32 (see 2.1 *BasicType*). The value is input in the integer representation. The integer representation is calculated using the exponent defined for the variable (see section 2.1 *Exp*): Value in integer representation = $10^{-Exp} *$ value 2. simpleType = float, double (see 2.1 *BasicType*). The value is input in the integer representation. The integer representation is calculated using the precision defined for the variable:

Format value	Description
	Value in integer representation = 10 $^{-Precision}$ * value 3. All other data types (see 2.1 *BasicType*). The value is input in the native representation (native representation see section 2.1 *BasicType*)
Empty("") or "native"	The value is input in the native representation (native representation see section 2.1 *BasicType*)
Format string	The value is input according to the specified formatting (**not currently implemented**).

If the format attribute is missing, the value is input in the same way as for the "native" format value.

Response:
`[<Set/>]`

2.4 Arrays

If the variables are created as an array in the external database ("no" attribute), e.g.

```
<Variable_0 _="B" ind="22" no="8" acc="RW"/>
```

the individual values for the arrays can be addressed by a variable suffix in square brackets:

2.4.1 Reading arrays

The addressed element in the array is attached in square brackets,

```
[<Get _="Vpath[element] "/>]
```

Vpath Path to address the array.
element Addressed array element.

e.g. the third value in the array:
`[<Get _="/Process/COM?/D?/Variable_0[3] "/>]`

If no suffix is specified, all elements are separated by commas when they are listed in the output:
`[<Get _="Vpath" >]`

Response:
`[<Get _="Value1,Value2,Value3,…Value8" />]`

2.4.2 Writing arrays

The addressed element in the array is attached in square brackets,

```
[<Set _="Vpath[element]" value="Value" />]
```

Vpath Path to address the array.
element Addressed array element.
Value Element value.

e.g. the third value in the array:
`[<Set _="/Process/Bus?/D?/Variable_0[3]" value="20" />]`

If no suffix is specified, all elements must be separated by commas when they are listed in the value. Gaps are not permitted:

```
[<Set _="Vpath"
value="Value1,Value2,Value3,Value4,Value5,Value6,Value7,Value8" />]
```

Response:
```
[<Set/>]
```

2.5 Error states when processing device variables

The variables defined in the `external` database can also save different error states depending on the device type or bus system. Particularly when querying via a communication protocol, communication errors or protocol errors can occur for example, so that the variable value (value from the last error-free access or the initial value) is invalid.

The error state for the variables is indicated by the two error codes ErrorClass and ErrorNumber. Both can be read out or processed further to generate an alarm.

2.5.1 Reading the error state out

If the value for a variable is invalid, i.e. an error has been detected and saved in the error state, this has the following effect on the "Get" read command.

1. TiXML "Get" command to read out the **variable group** (e.g. `[<Get _="/Process/COM2/D2"/>]):`
   Parameter entries with invalid values are **not listed** in the response.
2. TiXML "Get" command to read out **one variable**:
   (e.g. `[<Get _="/Process/COM2/D2/Sprache" />])`

   A TiXML error is issued in the response (see TiXML error frame in the TiXML reference manual):
   ErrorText „Variable exists but does not contain data"
   ErrNo -2194

   Note:
   Both commands provide the last value that was read without errors instead of an error notification if at least one access after uploading the variable configuration or a system start (reset, power on) was error-free.

In order to read out the error state of a parameter or parameter groups directly, an XML attribute is added to the TiXML "Get" command (see TiXML reference manual), which causes the "Get" command to output the value of an additional piece of information for a parameter, the error state in this case, instead of the parameter value:

Command:
```
[<Get _="VPath" AddInfo="AddInfo" />]
```

Overview of possible parameters (values written in italics):

Vpath Path to address the parameter.

AddInfo **Error .....** Returns the value's error state.

The "AddInfo" attribute is ignored for values that are not parameters for external devices. The parameter's value is returned.

Response:
AddInfo = Error
`[<Get _="ErrorClass,ErrorValue" />]`

ErrorClass: Error class

0	No error
>0	Error

ErrorValue: Error value

ErrorClass	ErrorValue	Meaning
0	0	No error
1	*n* > 0	Error in the FP IoT gateway's access implementation. The number *n* is > 0 and depends on the corresponding device or bus system.
c >1	*n*	Error message for the device that is connected or the bus protocol. The numbers *n* and *c* depend on the corresponding device or bus system.

Note:
The error state is deleted again (i.e. set to 0.0) by a subsequent error-free access to the PLC or the bus.

2.5.2 Processing the error state

In order to use the error state to generate alarms (or general events), a special loading command is defined in the instruction list for the "<Value/>" entry when defining a process variable (see the TiXML reference manual):

Command:
`<LDS _="Vpath" AddInfo="AddInfo" />`

Overview of possible parameters (values written in italics):

Description:
Reads out the error state of the parameter that is referenced in *Vpath* and writes the following values to the processing stack:

High part (bit 16 - 31)	Low part (bit 0 - 15)
Error class value	Error value

Vpath: Path to address the parameter.

AddInfo:

ErrorCode ..... Returns the value's error state.

3 Supported PLC systems

3.1 Mitsubishi Alpha XL

The variables to be monitored for the Mitsubishi Alpha XL that is connected must be defined in the FP IoT gateway.
The Mitsubishi variables are saved in the external group on the 'PROCCFG' database:

Use COM port COM2

```
<External>
   <Bus _="COM2" protocol="Mitsubishi,Alpha2" type="Master"
    baud="9600">
      <Device _=”0” Pollrate=”1s”>
         <Input1 _="I" ind="1"/>
         <ExtInput129 _="EI" ind="129" acc="R"/>
         <M1 _="M" ind="1" acc="R"/>
         <Keypad1 _="K" ind="1" acc="R" />
         <Display _="N" ind="3" acc="R" />
         <InAnalog7 _="AI" ind="7" acc="R" />
         <Counter1  _="CB" ind="1" acc="R" />
         <CW2 _="CW" ind="2" />
         <Out2 _="O" ind="2" acc="W"/>
         <ExtOut129 _="EO" ind="129" acc="W"/>
      </Device>
   </Bus>
</External>
```

Master communication

Variables list

The BUS parameter contains the address for the extension card, the protocol manufacturer “Mitsubishi”, the type of controller connected “Alpha2”, the “Master” communication mode and the baud rate used.

The device ID (station name) must match the PLC address (default 0).

If the FP IoT gateway is connected to the “MB” interface (main board), PLC communication starts directly after removing the PC cable and stops automatically when a TiXML command is detected.

A list of variables can be defined:

Variable type in the Alpha-XL (I,O,AI, etc.)

```
<Alarm11 _="I" ind="5" acc="R"/>
```

Each line defines a logical name (alias, e.g. Alarm11) and the variable type in the Mitsubishi PLC (see: list of supported variables)
The ‘ind’ parameter determines the variable’s address in the Mitsubishi controller and the ‘acc’ parameter the access right. The access right can either be ‘R’ or ‘RW’ for read or read/write access depending on the variable selected.

The ‘def’ parameter determines the start value for the variable. A variable with write access contains this start value until the first write access. A variable with read access contains this value until the device has received the actual value from the PLC.

List of supported variables for Alpha XL:

Type	index	access	Comment
M	1-14	R	System bits
I	1-15	R	Digital inputs
EI	129-132	R	Expansion module inputs
O	1-9	RW	Outputs
EO	129-132	RW	Expansion module outputs
K	1-8	R	BKeys
E	1-4	R	Link for inputs
A	1-4	RW	Link for outputs
N	1-4	RW	Control bits
AI	1-8	R	Analogue input
CB	1-100	RW	Bit operands
CW	1-100	RW	Word operands

If only read access 'R' is possible, the 'acc' parameter can be omitted.
If the PLC is in "RUN", some variables can be overwritten by the PLC program. In this case, markers must be connected upstream.

Note: If the automatically generated DeviceState (see section 6.3) is "1" but no variable values have been received, bit or word operands exist in the device or the Alpha XL, which are not defined on one of the two sides. These variables must always be present on both sides as otherwise, the Alpha XL does not respond.

Connecting an Alpha XL

The FP IoT gateway must be connected to the Alpha XL via a Mitsubishi GSM-CAB.

Note the following information:

1. In the Alpha XL, a program with activated "serial communication" must be present on 9600/8N1. (see the Alpha Programming Software online help). After activation, the Alpha must be restarted.
2. The GSM-CAB can be connected directly to the main board RS232 interface (MB).
3. If you connect the GSM-CAB to the FP IoT gateway extension card RS232-2 (C1), you must use a null modem cable between the device and the GSM-CAB.

Remote access

The following "TransMode" command is required for remote access to the PLC: (see the TiXML reference manual for further information)

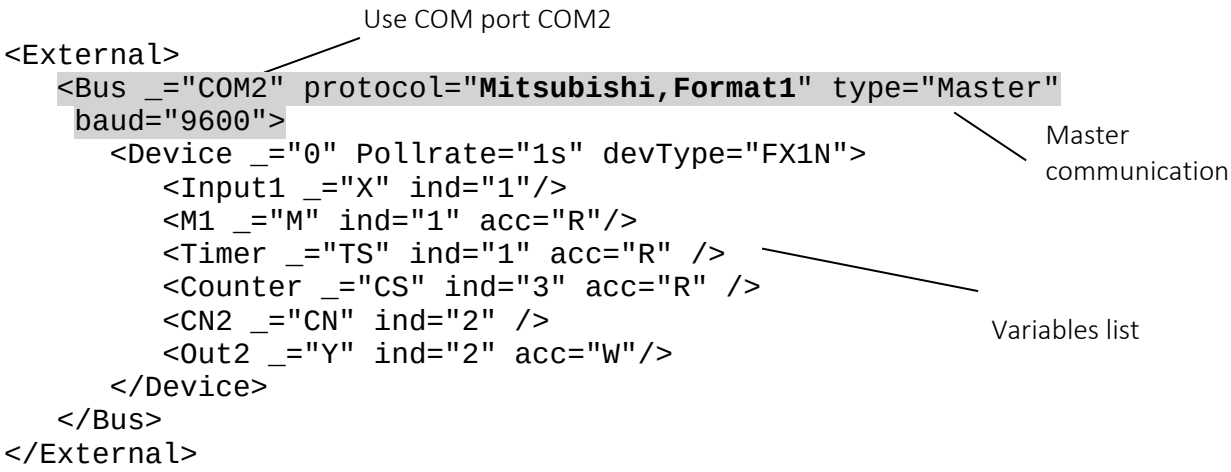
Alpha XL:

```
[<TransMode baud="9600" format="8N1" handshake=”noDTR” com="COM1"/>]
```

(use `com=”COM2”,` if the Alpha XL is connected to the COM2 RS232)

3.2 Mitsubishi MELSEC FX

The variables to be monitored for the Mitsubishi FX controller that is connected must be defined in the FP IoT gateway.
The Mitsubishi variables are saved in the external group on the 'PROCCFG' database:



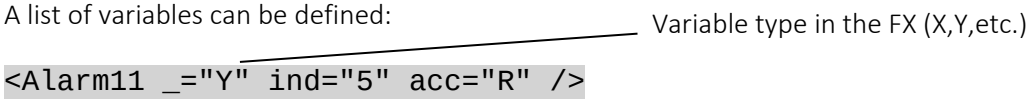

```
<External>
   <Bus _="COM2" protocol="Mitsubishi,Format1" type="Master"
    baud="9600">
      <Device _="0" Pollrate="1s" devType="FX1N">
         <Input1 _="X" ind="1"/>
         <M1 _="M" ind="1" acc="R"/>
         <Timer _="TS" ind="1" acc="R" />
         <Counter _="CS" ind="3" acc="R" />
         <CN2 _="CN" ind="2" />
         <Out2 _="Y" ind="2" acc="W"/>
      </Device>
   </Bus>
</External>
```

The BUS parameter contains the address for the extension card, the protocol manufacturer "Mitsubishi", the "Format1" protocol type, the "Master" communication mode and the baud rate used.

The device ID (station name) must match the PLC address (default 0).
The Format1 protocol is network-compatible (via RS485), whereby several devices with different IDs can exist.
The "devType" parameter determines the CPU type: FX1S,FX1N or FX2N.

If the FP IoT gateway is connected to the "MB" interface (main board), PLC communication starts directly after removing the PC cable and stops automatically when a TiXML command is detected.

A list of variables can be defined:




```
<Alarm11 _="Y" ind="5" acc="R" />
```

Each line defines a logical name (alias, e.g. Alarm11) and the variable type in the Mitsubishi PLC (see: list of supported variables)
The 'ind' parameter determines the variable's address in the Mitsubishi controller and the 'acc' parameter the access right. The access right can either be 'R' or 'RW' for read or read/write access depending on the variable selected.

The 'def' parameter determines the start value for the variable. A variable with write access contains this start value until the first write access. A variable with read access contains this value until the device has received the actual value from the PLC.

List of supported variables for MELSEC FX "Format1":

Type	index	access	Comment
M	0-8255	RW	Marker
Y	0-377	RW	Outputs (octal)
X	0-377	R	Inputs (octal)
S	0-999	RW	Step status
CS	0-255	RW	Meter contact
TS	0-255	RW	Timer contact

Type	index	access	Comment
TN	0-255	RW	Timer actual value, unsigned
TNI	0-255	RW	Timer actual value, signed
CN	0-199	RW	Meter actual value (16bit)
CD	200-255	RW	Meter actual value (32bit), unsigned
CDI	200-255	RW	Meter actual value (32bit), signed
D	0-8255	RW	Register (16 bit), unsigned
DI	0-8255	RW	Register (16 bit), signed
DW	0-8254	RW	Register (32 bit), unsigned
DWI	0-8254	RW	Register (32 bit), signed

If only read access 'R' is possible, the 'acc' parameter can be omitted.
If the PLC is in "RUN", some variables can be overwritten by the PLC program. In this case, markers must be connected upstream.

In addition to the open "Format1" protocol, the FX-internal protocol is integrated into the devices. Selection is via the bus parameter
`protocol="Mitsubishi,FX".` This protocol is not network-compatible but does provide additional access to the contacts and resets for the timer/counter:

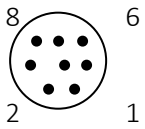
List of additional supported variables for MELSEC FX "FX protocol":

Type	index	access	Comment
CC	0-255	R	Meter spool
TC	0-255	R	Timer spool
CR	0-255	R	Meter reset
TR	0-255	R	Timer reset

Connecting a Mitsubishi FX
The FP IoT gateway can be connected to the FX internal RS422 interface or via an additional RS232-BD / RS422-BD / RS485-BD interface expansion.
If you use a BD expansion, this interface must be activated using the GX Developer Software with the parameters 9600/7E1.
Both interfaces can be used simultaneously, e.g. to connect a device and a display to the FX simultaneously.

RS422 connection:

FP IoT gateway (RS422 extension card)		MELSEC FX port (8pin female)
T+	----------	2
T -	----------	1
R-	----------	4
R+	----------	7



RS485 connection:

FP IoT gateway (RS485 extension card)		MELSEC FX RS485-BD (5pin screw terminal)
T+	----------	RDA
T -	----------	RDB
R-	----------	SDB
R+	----------	SDA
0V	----------	SG

Remote access

The following “TransMode” command is required for remote access to the PLC: (see the TiXML reference manual for further information)

MELSEC FX:

FP IoT gateway RS232 interface:

```
[<TransMode baud="9600" format="7E1" com="COM1"/>]
```

(use `com=”COM2”,` if the PLC is plugged into the COM2 RS232.)

FP IoT gateway RS422 interface:

```
[<TransMode baud="9600" format="7E1" handshake=”FULL” com="COM1"/>]
```

3.3 Siemens Simatic S7-200 via MPI interface

The variables to be monitored for the Siemens Simatic S7-200 that is connected must be defined in the FP IoT gateway.

The Siemens variables are saved in the external group on the 'PROCCFG' database:

```
<External>
   <Bus _=”COM2” protocol=”Siemens,S7-200” type=”Master” baud=”9600”
    handshake=”HALF” TS=”0” MAXADR=”15” GUF=”1” RC=”1”>
      <Device _=”2” Pollrate=”1s”>
         <M30 _="M" ind="30" acc="R" />
      </Device>
      <Device _=”3” Pollrate=”60s”>
         <V100 _="V" ind="100" acc="R" />
         <VS50 _="VS" size=”4” ind="100" acc="R" />
      </Device>
   </Bus>
</External>
```

RS485 interface on COM2

Station #2

Station #3

The BUS parameter contains the address for the COM port, the protocol manufacturer “Siemens”, the type of controller connected “S7-200”, the “Master” communication mode, the baud rate used and the required handshake (when using the RS485 extension card). TS is the device’s station number, MAXADR is the range of station numbers to be queried, GUF is the “gap update factor” to detect other slaves and RC is the number of repetitions in the event of communication errors.

The FP IoT gateway can only be the master to the S7-200. The default baud rate is 9600.

If the FP IoT gateway is connected to the “MB” interface (main board), PLC communication starts directly after removing the PC cable and stops automatically when a TiXML command is detected.

A ‘Device’ section that contains the station number (‘_’ – attribute) and the query cycle has to be inserted for each controller. The variable values are read in again after each query cycle.

A list of variables can be defined:

```
<AlarmM10 _="M" ind=”10” acc=”R”/>
```

Variable type in the S7-200

Each line defines a logical name (alias, e.g. Alarm11) and the variable type in the Siemens PLC (see: list of supported variables).

The ‘ind’ parameter determines the variable’s address in the Simatic controller and the ‘acc’ parameter the access right. The access right can either be ‘R’ or ‘RW’ for read or read/write access depending on the variable selected.
Access right ‘A’ must also be added, which enables access to the variables.

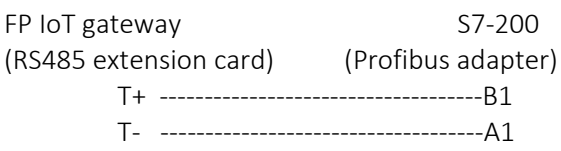
The ‘def’ parameter determines the start value for the variable. A variable with write access contains this start value until the first write access. A variable with read access contains this value until the device has received the actual value from the PLC.

List of supported variables for S7-200 CPUs

Type	index	access	Comment
V	0-10239.7	R/W	Variable memory (bit)
VB	0-10239	R/W	Variable memory (byte)
VW	0-10238	R/W	Variable memory (word)
VD	0-10236	R/W	Variable memory (double word)
VS	0-10239	R/W	Variable memory (string, requires “size” parameter)
I	0-15.7	R	Inputs
Q	0-15.7	R/W	Outputs
M	0-31.7	R/W	Marker (bit)
MB	0-31	R/W	Marker (byte)
MW	0-30	R/W	Marker (word)
MD	0-28	R/W	Marker (double word)
SM	0-549.7	R	Special marker
S	0-31.7	R/W	Sequence control relay
T	0-255	R/W	Timer
C	0-255	R/W	Meter
AI	0-62	R	Analogue input
AQ	0-62	R/W	Analogue output
HC	0-5	R	High speed meter

Connecting an S7-200

The S7-200 can be connected via a PPI cable (RS232) or via a Profibus adapter (RS485 extension card).



Remote access

The following “TransMode” command is required for remote access to the PLC: (see the TiXML reference manual for further information)

FP IoT gateway RS232 interface:

```
[<TransMode baud="9600" format="8E1" handshake="noDTR" com="COM1"/>]
```

(use `com=”COM1”,` if the PLC is plugged into the COM1 (RS232).)

FP IoT gateway RS485 interface:

```
[<TransMode baud="9600" format="8E1" handshake="HALF" com="COM1"/>]
```

In order to connect to an S7-200, an 11-bit modem (8E1 data format) must be used locally. Furthermore, the timings must be adjusted in the MicroWin software using a registry patch that was supplied on the Tixi CD.

3.4 Siemens Simatic S7-300/400 via MPI interface

The variables to be monitored for the Siemens Simatic S7-300/400 that is connected must be defined in the FP IoT gateway.
The Siemens variables are saved in the external group on the 'PROCCFG' database:

```
<External>
    <Bus Name="Bus1" _="COM2" family="Siemens" protocol="Siemens,S7-300/400-A"
     type="Master" TS="0" MAXADR="15" GUF="1" RC="1">

        <Device _="2" Name="Device_2" Pollrate="2s">
          <AlarmS7300 _="M"  ind="0.0" acc="RA"  def="0" format="?Alarm,OK"/>
          <Confirmation _="M"  ind="0.1" acc="RWA" def="1"/>
          <I124 _="VM" ind="0" db="15" acc="RA" def="0"/>
          <Test _="M" ind="0.3" acc="RA" def="0" format="?Alarm,OK"/>
        </Device>
        <Device _="3" Name="Device_3" Pollrate="60s">
          <AlarmS7300 _="M" ind="0.0" acc="RA" def="0" format="?Alarm,OK"/>
          <Confirmation _="M" ind="0.1" acc="RWA" def="1"/>
        </Device>
    </Bus>
</External>
```

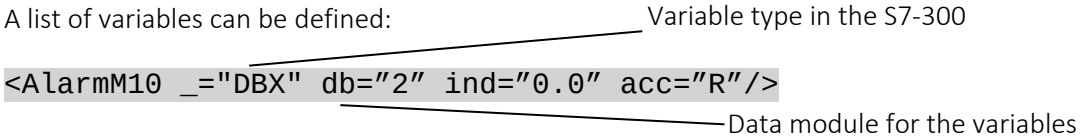
The BUS parameter contains the address for the COM port, the protocol manufacturer “Siemens”, the type of controller connected “S7-300/400-A” and the “Master” or “Slave” communication mode. TS is the device’s station number, MAXADR is the range of station numbers to be queried, GUF is the “gap update factor” to detect other slaves and RC is the number of repetitions in the event of communication errors.

When loading an external definition, the parameters stored in the TS adapter by the Siemens Teleservice software are deactivated.

The FP IoT gateway can be the master or the slave to the S7-300/400.

A ‘Device’ section that contains the station number (‘_’ – attribute) and the query cycle has to be inserted for each controller. The variable values are read in again after each query cycle (master) or a communication timeout detected (slave).

A list of variables can be defined:



Each line defines a logical name (alias, e.g. AlarmM10) and the variable type in the Siemens PLC (see: list of supported variables).
The ‘ind’ parameter determines the variable’s address in the Simatic controller and the ‘acc’ parameter the access right. The access right can either be ‘R’ or ‘RW’ for read or read/write access depending on the variable selected.

The ‘db’ parameter determines the data block number in which the variable is located. The value can be between 1 and 65535.

The ‘def’ parameter determines the start value for the variable. A variable with write access contains this start value until the first write access. A variable with read access contains this value until the device has received the actual value from the PLC.

List of supported variables for S7-300/400 CPUs

Type	index	access	Comment
DBX	0.0-65535.7	R/W	Data module (bit, requires “db” parameter)
DBB	0-65535	R/W	Data module (byte, requires “db” parameter)
DBW	0-65534	R/W	Data module (word, requires “db” parameter)
DBD	0-65532	R/W	Data module (double word, requires “db” parameter)
DBS	0-65535	R/W	Data module (string, requires “size” and “db” parameters)
I	0-16384.7	R	Inputs
Q	0-16384.7	R/W	Outputs
M	0-65535.7	R/W	Marker bit
MB	0-65535	RW	Marker byte
MW	0-65534	RW	Marker word
MD	0-65532	RW	Marker DWord
T	0-2074	R/W	Timer
C	0-2074	R/W	Meter

Connecting an S7-300/400

The S7-300/400 is connected to the FP IoT gateway via MPI using a Profibus adapter.

FP IoT gateway S7-300/400
(Profibus adapter) (Profibus adapter)
B1 ----------B1
A1 ----------A1

Remote access

Remote access is performed completely transparently using the TS adapter function.

The following conditional apply to TS adapter access protection:

1. Local access to the S7 via the device (COM1->COM2) cannot be blocked. (It therefore behaves in the same way as the original Siemens TS adapter).

2. The TS users that are set using the SIMATIC software (“Set TS adapter parameters” menu) work in the same way as for the original Siemens TS adapter, i.e. an ADMIN and 2 users can be created, as well as a call-back function for the users.

3. The TS adapter settings are deleted by a factory reset.

4. As soon as an S7 connection (external) AND an AccRights user (see TiXML reference manual) have been defined in the device, only the users created in AccRights with the "TSAdapter" service are valid. A call-back can also be created for these using the user attribute with the same name. The users set in the TS adapter are retained but are inactive. These only become valid again after deleting the AccRights.

Additional benefit: Virtually any number of users can be created in the AccRights, the limitation of 3 users (as in the TS adapter) does not apply here.

Example of TS adapter AccRights:

```
[<SetConfig _="USER" ver="y">
   <AccRights>
      <Groups>
         <Custodian>
            <TSAdapter AccLevel="1"/>
         </Custodian>
      </Groups>
      <User>
         <Smith Plain="Dachshund" Group="Custodian" Callback="123"/>
      </User>
   </AccRights>
</SetConfig>]
```

3.5 Siemens Simatic S7-200/300/400/1200/1500 via LAN interface

The S7 controllers communicate via LAN (TCP/IP) using the "ISO-ON-TCP" protocol RFC1006 (sometimes also known as the "AGLink" protocol). Siemens S7-200, S7-300/400, S7-1200, S7-1500 are currently supported.
Configuration is basically identical for the individual PLC types.

3.5.1 General information and notes

Currently (firmware version 3.04.01.016), a reset must be performed each time after loading a new configuration (PROCCFG). In the bus and device parameters, uppercase and lowercase are differentiated between, "PLCC" is therefore not the same as "plcc".

3.5.2 Bus parameters

```
<Bus _="ETH" BusId="1" name="S7_300" protocol="Siemens,AGLink" type="Master">
```

_="ETH"	Designates the Ethernet interface for the FP IoT gateway (on the main board)
"BusId"	Enables the PLC bus to be addressed regardless of the interface. Must not be confused with "Name".
"Name"	Enables the PLC bus to be named and therefore enables addressing regardless of the interface. Must not be confused with "BusId".
"protocol"	Protocol name, must be set to "Siemens,AGLink"
"type"	Protocol type, must always be set to "Master"

3.5.3 Device parameters

```
<Device _="2" Name="Dev2" Pollrate="1s" IP="193.101.167.203"
 GW="193.101.167.193" mask="255.255.255.0" rack="0" slot="2" PLCC="0">
```

"2"	FP IoT device number; corresponds to the connected CPU's PLC number
"IP"	IP address for the Siemens PLC
"GW"	Network gateway to be used
"mask"	Network mask for the IP address (default: 255.255.255.0)
"rack"	Rack number for the Siemens CPU (default: 0)
"slot"	Slot number for the Siemens CPU (default: 300:2, 400:3, 1200:1)
"PLCC"	PLC class. The type of controller connected. 0: 300/400, 1: 200, 2: 1200 (default: 1)

3.5.4 Further parameters for internal use only:

"`credit`" Required for certain external manufacturers (PLC), not for Siemens

"`ctype`" Type of communication unit that is connected. 0: PD, 1: OP, 2: other. (PD = programming device; OP = operator panel)

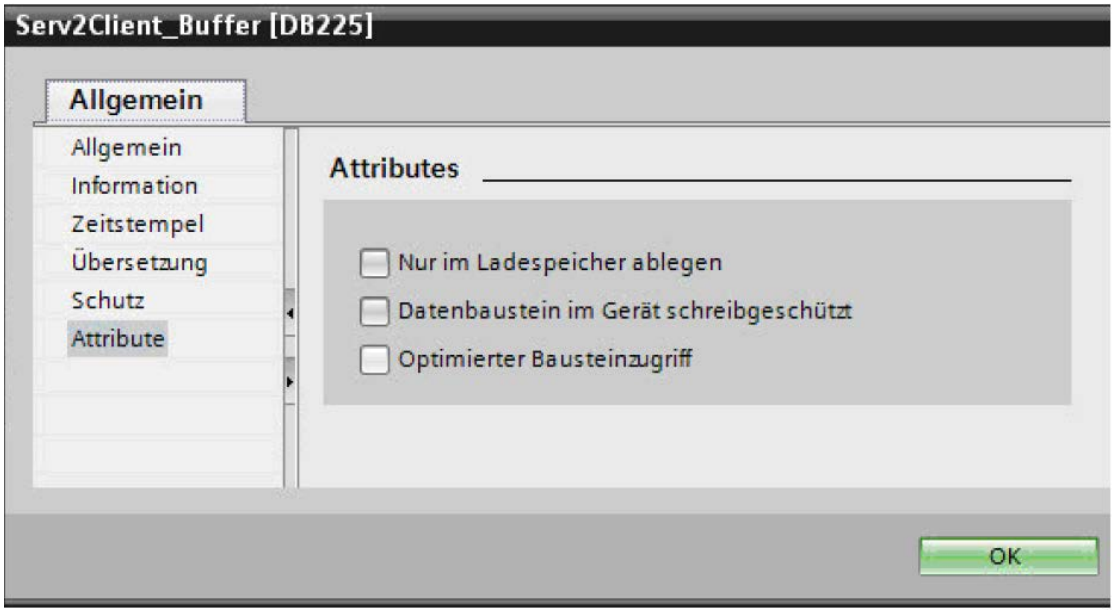
"`port`" The port number for TCP/IP traffic is 102 by default.
If a different port number is used in exceptional cases, this must be specified here.
Port 102 is always used for Siemens controllers.

"`Timeout`" Default timeout for communication. If there is no unit of measurement or if the "ms" (milliseconds) unit of measurement is used, the value is adopted as is (as ms). If the unit of measurement is a different valid unit of measurement, it is converted to ms accordingly. If the parameter is not specified or if the value range of a double word (32 bit) is exceeded, the default value 6000(ms) is set.
In most applications, this parameter does not need to be specified.

Special features for S7-1200 and S7-1500

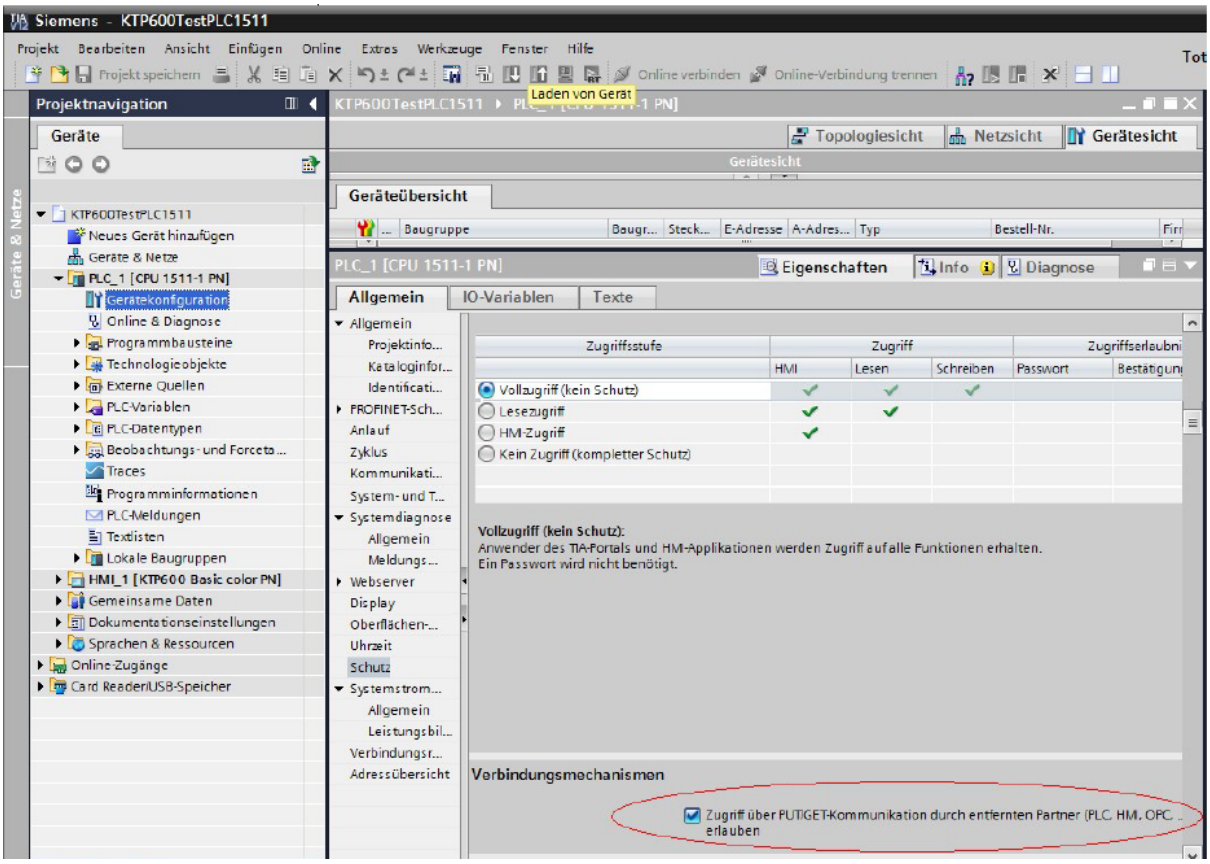
Access to S7-1200/1500 is basically possible and is similar to accessing an S7-300 or S7-400. However, the following points must be noted:

- Data modules may only be accessed with absolute addressing. Therefore, the "Optimised module access" (Optimierter Bausteinzugriff) attribute must be deactivated in the properties for the data modules in the PLC project.



Accessing peripheral device input and peripheral device output data on an S7-1200/1500 is not possible.

- Accessing timers (times), counters (meters) and 64 bit data types on an S7-1200/1500 is not possible.
- On the S7-1500 and the S7-1200 (as of firmware version 4), PUT/GET communication is essential and must be allowed. This must be done in the device configuration in the PLC project.



- No direct access is possible to the new data types that appeared with the S7-1200/1500.
  No access is currently available to 64 bit data types. It is possible that the new data type can be converted into a known 32 bit data type (REAL or DWORD) and be output in a different data range (e.g. as a marker, DB, etc.).
  In order to access the remaining new data types, a comparable data type from an S7-300/400 can be used.

The following new data types are available on the S7-1500 with partially comparable data types on the S7-300/400:

S7-1500 data type	Comparable S7-300/400 data type
USINT (8 bit)	BYTE
SINT (8 bit)	BYTE
UINT (16 bit)	WORD
UDINT (32 bit)	DWORD
ULINT (64 bit)	not currently possible
LINT (64 bit)	not currently possible
REAL (32 bit)	REAL
LREAL (64 bit)	not currently possible

3.5.5 Variables parameters

3.5.5.1 S7-300 / 400, S7-1200, S7-1500

Supported variables for S7-300 / 400 CPUs via LAN
(different values for S7-1200 / S7-1500 in brackets)

Type	index	access	Comment
DBX	0.0-65535.7	R/W	Data module (bit, requires “db” parameter)
DBB	0-65535	R/W	Data module (byte, requires “db” parameter)
DBW	0-65534	R/W	Data module (word, requires “db” parameter)
DBD	0-65532	R/W	Data module (double word, requires “db” parameter)
DBS	0-65535	R/W	Data module (string, requires “size” and “db” parameters) (S7-1200: Please use DBB!)
I	0.0-16384.7	R	Inputs (S7-1200: index 0.0 - 1023.7)
Q	0.0-16384.7	R/W	Outputs (S7-1200: index 0.0 - 1023.7)
M	0-65535.7	R/W	Marker bit
MB	0-65535	R/W	Marker byte
MW	0-65534	R/W	Marker word
MD	0-65532	R/W	Marker DWord
T	0-999	R/W	Timer
C	0-999	R/W	Meter

The access right can either be ‘R’ or ‘RW’ for read or read/write access depending on the variable selected. Access right ‘A’ must also be added, which enables access to the variables (i.e. `acc="RA"` for read, `acc="RWA"` for read + write).

The ‘db’ parameter determines the data block number in which the variable is located. The value can be between 1 and 65535.

Example

```
<AlarmM10 _="DBX" db="2" ind="0.0" acc="RA" />
```

"DBX" "Data module” variable type
"2" Data module for the variables
"0.0" Index for the variable within the data module
"RA" ”Read” access right

3.5.5.2 S7-200

Supported variables for S7-200 CPUs via LAN:

Type	index	access	Comment
DBX	0.0-10239.7	R/W	Corresponds to data type V (variable memory bit). Access via db=1: db="1" DBX="0.0 – 10239.7"
DBB	0-10239	R/W	Corresponds to data type VB (variable memory byte). Access via db=1 DBB="..": db="1" DBB="0 – 10239"
DBW	0-10238	R/W	Corresponds to data type VW (variable memory word). Access via db=1 DBW="..": db="1" DBW="0 – 10238"
DBD	0-10236	R/W	Corresponds to data type VD (variable memory Dword). Access via db=1 DBD="..": db="1" DBD="0 – 10236"
I	0.0-15.7	R	Inputs
Q	0.0-15.7	R/W	Outputs
M	0-31.7	R/W	Marker bit

Type	index	access	Comment
MB	0-31	R/W	Marker byte
MW	0-30	R/W	Marker word
MD	0-28	R/W	Marker Dword
M	256.0 – 805.7	R	Corresponds to data type SM (special marker bit). Access via M with offset 256 M="256.0" corresponds to SM="0.0", M="805.7" corresponds to SM="549.7"
T	0-255	R/W	Timer
C	0-255	R/W	Meter

3.5.6 Examples

External configuration for a Siemens S7-300

```
[<SetConfig _="PROCCFG" ver="v">
<External>
   <Bus _="ETH" BusId="1" Name="S7_300" protocol="Siemens,AGLink" type="Master">
      <Device _="2" Name="Dev2" Pollrate="1s"
        IP="193.101.167.203" GW="193.101.167.193"
        mask="255.255.255.0" rack="0" slot="2" PLCC="0">
         <MB0 _="MB" acc="RWA" def="" ind="0"/>
         <MB1 _="MB" acc="RWA" def="" ind="1"/>
         <MW2 _="MW" acc="RWA" def="" ind="2"/>
         <M0.0 _="M" acc="RWA" def="" ind="0.0"/>
         <M0.1 _="M" acc="RWA" def="" ind="0.1"/>
         <T0 _="T" acc="RWA" def="" ind="0"/>
         <C0 _="C" acc="RWA" def="" ind="0"/>
         <I0.0 _="I" acc="RA" def="" ind="0.0"/>
         <I0.1 _="I" acc="RA" def="" ind="0.1"/>
      </Device>
   </Bus>
</External>
</SetConfig>]
```

External configuration for a Siemens S7-1200

```
[<SetConfig _="PROCCFG" ver="v">
<External>
   <Bus _="ETH" BusId="1" Name="S7_1200" protocol="Siemens,AGLink" type="Master">

      <Device _="1" Pollrate="1s" IP="193.101.167.135" GW="193.101.167.193"
              mask="255.255.255.128" rack="0" slot="1" PLCC="2">

         <!--Inputs-->
         <I0.0 _="I" acc="RA" def="" ind="0.0"/>

         <!--Outputs-->
         <Q0.0 _="Q" acc="RA" def="" ind="0.0"/>

      </Device>
   </Bus>
</External>
</SetConfig>]
```

3.6 VIPA CPU 100/200/300

The variables to be monitored for the VIPA CPU that is connected must be defined in the FP IoT gateway.

The VIPA CPUs can of course also be connected to a device with an MPI interface. The range of functions is identical (see section 3.4)

The VIPA variables are saved in the external group on the 'PROCCFG' database:

Use COM port on COM2

```
<External>
   <Bus _=”COM2” protocol=”Vipa,GreenCable” type=”Master”
    baud=”38400” TS=”1”>
      <Device _=”2” Pollrate=”1s”>
         <M30 _="M" ind="30" acc="R" />
      </Device>
   </Bus>
</External>
```

Station #2

The BUS parameter contains the address for the “COM2” COM port, the protocol manufacturer “Vipa”, the “GreenCable” protocol, the “Master” communication mode, the baud rate “38400” and the TS parameter that contains the FP IoT gateway’s station number.

The FP IoT gateway is always the “master” to the VIPA.
The device ID (station name) must match the PLC address.

If the FP IoT gateway is connected to the “COM1”, PLC communication starts directly after removing the PC cable and stops automatically when a TiXML command is detected.

A ‘Device’ section that contains the station number (‘_’ – attribute) and the query cycle has to be inserted for each controller. The variable values are read in again after each query cycle (master) or a communication timeout detected (slave).

A list of variables can be defined:

Variable type in the VIPA

```
<AlarmM10 _="M" ind=”10” acc=”R”/>
```

Each line defines a logical name (alias, e.g. AlarmM10) and the variable type in the VIPA CPU (see: list of supported variables)

The ‘ind’ parameter determines the variable’s address in the Simatic controller and the ‘acc’ parameter the access right. The access right can either be ‘R’ or ‘RW’ for read or read/write access depending on the variable selected.

The ‘def’ parameter determines the start value for the variable. A variable with write access contains this start value until the first write access. A variable with read access contains this value until the device has received the actual value from the PLC.

List of supported variables for VIPA System 200V CPU

Type	index	access	Comment
M	0-2047.7	R/W	Marker (bit)
MB	0-2047	R/W	Marker (byte)
MW	0-2046	R/W	Marker (word)
MD	0-2044	R/W	Marker (double word)
I	0-15.7	R	Inputs
Q	0-15.7	R/W	Outputs

The FP IoT gateway also supports the VIPA System 100V CPU and the VIPA System 300V CPU.
The VIPA is only supported up to firmware version 2.0 for devices in the Aluline range.

Remote access
The following “TransMode” command is required for remote access to the PLC: (see the TiXML reference manual for further information)

```
[<TransMode baud="38400" format="8O1" com="COM1"/>]
```

(Use `com=”COM2”,` if the VIPA is connected to the COM2 RS232)

In order to establish a connection to a VIPA CPU, an 11-bit modem (support for 8O1 data format) must be used locally.

3.7 Moeller Easy 400 / 500 / 600 / 700 / 800 / MFD

The variables to be monitored for the Moeller Easy (e.g. easy 400/600/800/MFD) that is connected must be defined in the FP IoT gateway.
The Moeller variables are saved in the external group on the 'PROCCFG' database:

```
<External>
   <Bus _=”COM1” protocol=”Moeller,Easy 800” type=”Master”
    baud=”9600” handshake=”noDTR”>
      <Device _=”0” Pollrate=”1s”>
         <Input1 _="I" ind="1"/>
         <UIn _="AI" ind="7" acc="R" format="F.1;°C" def="1"/>
         <Out4 _="M" ind="2" acc="RW"/>
         <OutClock _="OU" ind="0" acc="R" />
         <OutCount _="OC" ind="0" acc="R" />
         <OutAnalog _="OA" ind="0" acc="R" />
         <OutTimer _="OT" ind="0" acc="R" />
         <Counter  _="IC" ind="0" acc="R" />
         <TextMarker _="D" ind="0" />
         <Timer _="IT" ind="0" />
         <Out2 _="Q" ind="2" acc="W"/>
         <Flag _="M" ind="8"/>
         <Clock _="U" ind="0" acc="RW"/>
      </Device>
   </Bus>
</External>
```

Use COM port on extension card C1

Easy 800 Master

Variables list

The BUS parameter contains the address for the “COM1” serial interface, the protocol manufacturer “Moeller”, the device connected “Easy 400/600”, “Easy 500/700“ or “Easy 800”, the “Master” communication mode, the baud rate and the required “noDTR” handshake.

For Easy 400//600, select “Moeller, Easy 400/600”, baud rate: 4800.
For Easy 500//700, select “Moeller, Easy 500/700”, baud rate: 4800.
For Easy 800/MFD, select “Moeller, Easy 800”, baud rate: 9600 or 19200.

Easy 400//600: The ‘Device’ entry must be “1” and contain the query cycle.
Easy 500/700/800/MFD: The ‘Device’ entry must contain the station name for the Easy that is connected. For a single Easy 500/700/800/MFD, this is always “0”. The Easy 800/MFD supports several stations (1-8).

If the FP IoT gateway is connected to the “MB” interface (main board), PLC communication starts directly after removing the PC cable and stops automatically when a TiXML command is detected.

A list of variables can be defined:

Variable type in the Easy

```
<Alarm11 _="I" ind=”16” acc=”R”/>
```

Each line defines a logical name (alias, e.g. Alarm11) and the variable type in the Moeller Easy (see: list of supported variables)
The ‘ind’ parameter determines the variable’s address in the Moeller Easy and the ‘acc’ parameter the access right. The access right can either be ‘R’ or ‘RW’ for read or read/write access depending on the variable selected.

The ‘def’ parameter determines the start value for the variable. A variable with write access contains this start value until the first write access. A variable with read access contains this value until the device has received the actual value from the Easy.

List of supported variables for Easy 400/600:

Type	index	access	Comment
IK	1-16	R	Expansion inputs
I	1-8	R	Inputs
OU	1-4	R	Time switch output
OC	1-8	R	Meter output
OA	1-8	R	Analogue value comparator output
OT	1-8	R	Timer output
IC	1-8	R	Meter actual value
D	1-8	RW	Text display
IT	1-8	R	Timer actual value
Q	1-8	R(W)	Outputs (write only in STOP)
M	1-16	RW	Marker
U	-	RW	Clock
ST	1-8	R	Timer target value
SC	1-8	R	Counter target value
AW	1-8	R	Analogue value comparator target value
QK	1-8	R(W)	Expansion outputs (write only in STOP)
Key	1-4	R	Test
AI	7.8	R	Analogue input

List of supported variables for Easy 500/700:

Type	index	access	Comment
I	1-16	R	Inputs
Q	1-8	R	Outputs
R	1-16	R	EASY-LINK input
S	1-8	R	EASY-LINK output
OU	1-8	R	Time switch output, as OUW (Easy 400/600 compatibility)
OUW	1-8	R	Weekly time switch output
OUY	1-8	R	Yearly time switch output
OC	1-16	R	Meter output
OA	1-16	R	Analogue value comparator output
OT	1-16	R	Timer output
IC	1-16	RW	Meter actual value
D	1-16	R	Text display
IT	1-16	RW	Timer actual value
IW	1-4	RW	Operating hours counter actual value
SW	1-4	RW	Operating hours counter target value
N	1-16	RW	Marker N

Type	index	access	Comment
M	1-16	RW	Marker M
U	-	RW	Clock
ST	1-16	RW	Timer target value 1
ST2	1-16	RW	Timer target value 2
SC	1-16	RW	Meter target value
AW	1-16	RW	Analogue value comparator target value, as AWI2 (Easy 400/600 compatibility)
AWI1	1-16	RW	Analogue value comparator target value 1
AWI2	1-16	RW	Analogue value comparator target value 2
AWF1	1-16	RW	Analogue value comparator amplifier
AWF2	1-16	RW	Analogue value comparator amplifier 2
AWOS	1-16	RW	Analogue value comparator offset
AWHY	1-16	RW	Analogue value comparator hysteresis
Key	1-4	R	Keys
AI	7,8.11,12	R	Analogue input

List of supported variables for Easy 800 / MFD:

Type	index	access	Comment
I	1-16	R	Inputs
Q	1-8	RW	Outputs (write only in STOP)
R	1-16	R	Expansion inputs
S	1-8	RW	Expansion outputs (write only in STOP)
M	1-96	RW	Marker
MB	1-96	RW	Byte marker
MW	1-96	RW	Word marker
MD	1-96	RW	Double word marker
Key	1-4	R	Keys
IA	1-4	R	Analogue inputs
QA	-	RW	Analogue outputs
U	-	RW	Clock

If only one address such as “U” exists for a variable, the ‘ind’ parameter can be omitted.
If only read access ‘R’ is possible, the ‘acc’ parameter can be omitted.
If the PLC is in “RUN”, some variables can be overwritten by the Easy program. In this case, markers must be connected upstream.

A **Type** variable is inserted automatically for each device for which parameters were set:
`[<Get _=”/Process/COM?/D?/Type”/>]`
This variable contains the type of Easy connected, e.g. “412-DC-Rx”.

Remote access
The following “TransMode” command is required for remote access to the Easy: (see the TiXML reference manual for further information)

Easy 400/500/600/700
`[<TransMode baud="4800" format="8N1" handshake=”noDTR” com="COM2"/>]`
(Use `com=”COM1”` if the PLC is plugged into the COM1 RS232.)

Easy 800/MFD
`[<TransMode baud="9600" format="8N1" handshake=”noDTR” com="COM2"/>]`

(Use `com=”COM1”` if the PLC is plugged into the COM1 RS232.)
In order to connect to an Easy via GSM, al least Easy-Soft 5.01 is required (waiting time can be adjusted).

3.8 Moeller PS30 & PS4/40

The variables to be monitored for the PS30 or PS4/40 that is connected must be defined in the FP IoT gateway.
The Moeller variables are saved in the external group on the 'PROCCFG' database:

```
<External>
   <Bus _=”COM1” protocol=”Moeller,SucomA” type=”Master”
    baud=”9600”>
      <Device _=”7” Pollrate=”1s”>
          <Value1 _="S" ind="1" acc=”R”/>
          <Alarmflag _="F" ind="7.0" acc="R"/>
          <Temp _="B" ind="2" acc="RW"/>
          <Power _="R" ind="127" acc="RW" />
          <CountVal _="D" ind="3212" acc="RW" />
      </Device>
   </Bus>
</External>
```

Use COM port COM1

SucomA master

Variables list

The BUS parameter contains the address for the “COM1” COM port, the protocol manufacturer “Moeller”, the “SucomA” protocol, the “Master” communication mode and the baud rate (4800 to 57600).

The station number must be “7“.

If the FP IoT gateway is connected to the “MB” interface (main board), PLC communication starts directly after removing the PC cable and stops automatically when a TiXML command is detected.

A list of variables can be defined:

```
<Alarm11 _="F" ind=”16” acc=”R”/>
```

Variable type in the PS30 & PS4/40

Each line defines a logical name (alias, e.g. Alarm11) and the variable type in the Moeller PLC (see: list of supported variables)
The ‘ind’ parameter determines the variable’s address in the Moeller PLC and the ‘acc’ parameter the access right. The access right can either be ‘R’ or ‘RW’ for read or read/write access depending on the variable selected.

The ‘def’ parameter determines the start value for the variable. A variable with write access contains this start value until the first write access. A variable with read access contains this value until the device has received the actual value from the PLC.

List of supported variables for PS30 & PS4/40:

Type	index	access	Comment
S	0-65535	R	Status WORD
F	0-14999.7	R	Marker BIT
B	0-14999	RW	Marker BYTE
R	0-14998	RW	Marker WORD
D	0-14996	RW	Marker DWORD

If only read access ‘R’ is possible, the ‘acc’ parameter can be omitted.
If the PLC is in “RUN”, some variables can be overwritten by the PLC program. In this case, markers must be connected upstream.

Remote access

The following “TransMode” command is required for remote access to the PLC: (see the TiXML reference manual for further information)

```
[<TransMode baud="9600" format="8N1" com="COM1"/>]
```

(Use `com=”COM2”,` if the PLC is connected to the COM2 RS232)

3.9 SAIA Burgess S-bus

The variables to be monitored for the SAIA controller that is connected (e.g. PCD2) must be defined in the FP IoT gateway.

The SAIA variables are saved in the external group on the 'PROCCFG' database:

```
<External>
   <Bus _=”COM2” protocol=”Saia,SBus-DataMode” type=”Master”
    baud=”19200” handshake=”HALF”>
      <Device _=”1” Pollrate=”1s”>
          <F100 _="F" ind="100" acc="R" />
      </Device>
      <Device _=”3” Pollrate=”60s”>
          <R102 _="R" ind="102" acc="R" />
      </Device>
   </Bus>
</External>
```

Use RS485 port on COM2

aktivates RS485 mode

S-Bus Station #1

S-Bus Station #3

The FP IoT gateway can be set as the S_BUS master or the slave. All S-BUS baud rates are supported, the standard baud rate (no entry) is 19200.

If the FP IoT gateway is connected to the “MB” interface (main board), PLC communication starts directly after removing the PC cable and stops automatically when a TiXML command is detected.

The `handshake=”HALF”` parameter activates RS485 mode on special extension cards.

A ‘Device’ entry must be created for each S-bus station, which contains the station number for the slave-PCD2s that are connected and its own station number as the S-BUS slave (‘_’ – attribute), as well as the query cycle.

The variable values are read in again once the query cycle elapses (master) or a communication timeout is detected (slave).

A list of variables can be defined:

```
<Alarm11 _="F" ind=”11” acc=”R”/>
```

Variable type in S-BUS (F,T,C,I,O,R)

Each line defines a logical name (alias, e.g. Alarm11) and the variable type on the S-BUS (see: list of supported variables)

The ‘ind’ parameter determines the variable’s address in the PCD2 and the ‘acc’ parameter the access right. The access right can either be ‘R’ or ‘RW’ for read or read/write access depending on the variable selected.

An additional access right is ‘RWL’ that activates a joint read and write memory that may only be used in slave mode. With the ‘RW’ access right, the FP IoT gateway has a separate memory for write and read accesses. In this case, the PLC can write a variable value to the FP IoT gateway’s read memory and the FP IoT gateway writes the same variable to the write memory. Therefore, the situation may arise in which the PLC writes a ‘1’ for example but receives a 0 when reading out subsequently.

With ‘RWL’, the FP IoT gateway uses the same memory for both accesses, i.e. the last value written always applies regardless of who set it.

The ‘def’ parameter determines the start value for the variable. A variable with write access contains this start value until the first write access. A variable with read access contains this value until the device has received the actual value from the PLC.

List of supported variables for PCD2

Type	index	access	Comment
F	0-8191	R/W	Flag
R	0-4095	R/W	Register
T	0-1600	R	Timer
C	0-1600	R	Counter
O	0-256	R/W	Output
I	0-256	R	Input (only as master)

Connecting the PCD2

The FP IoT gateway can be connected to the PCD2 on all 3 serial interfaces S0-S2. Only a 3-wire line (RX,TX,GND) is required.

Note the following information:

1. If you connect the FP IoT gateway to the PGU port (S0) on the PCD2, the DSR line is not permitted to be carried along, as the PCD2 would otherwise deactivate the S-BUS.

2. If the FP IoT gateway is connected to the RS232 main board (MB) on the PCD2, the DTR line is not permitted to be carried along, as the S-BUS would otherwise be deactivated.

Tixi-MB	PCD2	Tixi-Cx	PCD2	Tixi-RS485	PCD2
2-RxD	RxD-12/32	2-RxD	TxD-12/32	1-GND	GND-10/35
3-TxD	TxD-11/31	3-TxD	RxD-11/31	4-R(-)	RX-TX-11/36
5-GND	GND-10/30	5-GND	GND-10/30	5-R(+)	/RX-/TX-12/37

Remote access

The following “TransMode” command is required for remote access to the PLC: (see the TiXML reference manual for further information)

FP IoT gateway RS232 interface:

```
[<TransMode baud="9600" format="8N1" com="COM1"/>]
```

(Use `com=”COM2”,` if the FP IoT gateway is connected to the COM2 RS232)

FP IoT gateway RS485 interface:

```
[<TransMode baud="9600" format="8N1" handshake=”HALF” com="COM1"/>]
```

Select the same baud rate as for the PLC.

3.10 Carel Macroplus

The variables to be monitored for the Carel controller that is connected (e.g. Macroplus) must be defined in the FP IoT gateway.
The Carel variables are saved in the external group on the 'PROCCFG' database:

```
<External>
   <Bus _="COM2" protocol="Carel,PC2" type="Master"
    handshake="FULL">
      <Device _=”1” Pollrate=”1s”>
         <Alarm11 _="D" ind="22" acc="RW"/>
      </Device>
      <Device _=”3” Pollrate=”60s”>
         <Alarm31 _="D" ind="22" acc="RW"/>
      </Device>
   </Bus>
</External >
```

Use RS422 port on COM2

Aktivates RS422 mode

CarelBus controller #1

CarelBus controller #3

The `handshake=”FULL”` parameter activates RS422 mode. RS232 communication is used if this entry is not present.

If the FP IoT gateway is connected to the “MB” interface (main board), PLC communication starts directly after removing the PC cable and stops automatically when a TiXML command is detected.

A ‘Device’ entry that contains the station number on the Carel bus (‘_’ – attribute) and the query cycle has to be inserted for each controller located on the Carel bus. The FP IoT gateway queries the controller for changed variables as long as it does not receive a NULL frame that informs the device that no further changes are present.

A list of variables can be defined:

```
<Alarm11 _="D" ind=”22” acc=”RW”/>
```

Variable type in the Carel PLC (D,I,A)

Each line defines a logical name (alias, e.g. Alarm11) and the variable type in the Carel controller (see: list of supported variables)
The ‘ind’ parameter determines the variable’s address in the Carel controller and the ‘acc’ parameter the access right. The access right can either be ‘R’ or ‘RW’ for read or read/write access depending on the variable selected.

The ‘def’ parameter determines the start value for the variable. A variable with write access contains this start value until the first write access. A variable with read access contains this value until the device has received the actual value from the PLC.

List of supported variables for Macroplus

Type	index	access	Comment
D	1-183	R/W	Bits
I	1-50	R/W	Integer
A	1-50	R/W	Analogue

Connecting the Macroplus

The Macroplus can be connected via an RS422-RS232 adapter or directly to an RS422 extension card.

FP IoT gateway (RS422 extension card) — Macroplus (9pin RS422 female)

```
T+ ----------R+ (4)
T- ----------R- (5)
R- ----------T- (1)
R+ ----------T+ (2)
```

Remote access

The following “TransMode” command is required for remote access to the PLC: (see the TiXML reference manual for further information)

FP IoT gateway RS232 interface:

```
[<TransMode baud="1200" format="8N2" com="COM1"/>]
```

(Use `com=”COM2”,` if the FP IoT gateway is connected to the COM2 RS232)

FP IoT gateway RS422 interface:

```
[<TransMode baud="1200" format="8N2" handshake=”FULL” com="COM1"/>]
```

3.11 ABB AC010, AC031, CL series

The AC010 logic controller is an OEM version of the Moeller EASY 400/600 products.
See section 3.7 for details.

The AC031 is supported by the MODBUS RTU protocol.
See section 4.3 for details.

The CL series is an OEM version of the Moeller EASY 500/700 products.
See section 3.7 for details.

3.12 Allen Bradley Pico

This series of intelligent small control relays is an OEM version of the Moeller small controllers.
Pico series A = Moeller Easy 400/600
Pico series B = Moeller Easy 500/700
Pico GFX = Moeller MFD

See section 3.7 for details.

3.13 Theben PHARAO II

The PHARAO II small controller is an OEM version of the Mitsubishi Alpha2 (XL).
See section 3.1 for details.

4 Field bus support

4.1 Tixi-Bus

Tixi-Bus is a simple field bus protocol to exchange variables effectively with several PLC systems.
(See the InovoLabs website for further information or send an e-mail to tixi-support@inovolabs.com)

The Tixi-Bus variables are defined in the external group on the 'PROCCFG' database.

```
<External>
   <Bus _="COM2" protocol="Tixi.Com,Tixi-Bus" type="Master"
    handshake="none" TS="0">
      <Device _="1" Pollrate="1s">
         <Alarm11 _="F" ind="22" acc="RW"/>
      </Device>
      <Device _="3" Pollrate="60s">
         <Alarm31 _="R" ind="243" acc="RW"/>
         <Array _="B" ind="1" no="8" acc="RW"/>
      </Device>
   </Bus>
</External>
```

Use RS422 interface on COM2

PLC #1

PLC #3

The device can only act as a Tixi-Bus "master".

The `handshake="mode"` parameter activates various communication modes:
No entry is required for an RS232 interface.
When handshake="HALF", RS485 communication is used and when "FULL", RS 422 communication is used. A special extension card is required for RS 485/422.

TS is the device's station number.

If the FP IoT gateway is connected to the "MB" interface (main board), PLC communication starts directly after removing the PC cable and stops automatically when a TiXML command is detected.

A 'Device' entry that contains the station number for the connected slave PLC systems and the query cycle has to be inserted for each TixiBus station. The variable values are read in again after the query cycle elapses.

A list of variables can be defined:

```
<Alarm11 _="F" ind="22" acc="RW"/>
```

Variable type in the TixiBus protocol (F,W,etc.)

Each line defines a logical name (alias, e.g. Alarm11) and the variable type in the controller (see: list of supported variables)
The 'ind' parameter determines the variable's address in the controller and the 'acc' parameter the access right. The access right can either be 'R' or 'RW' for read or read/write access depending on the variable selected.

The 'def' parameter determines the start value for the variable. A variable with write access contains this start value until the first write access. A variable with read access contains this value until the device has received the actual value from the PLC.

In Tixi-Bus, variables can be created as arrays using the “no” attribute.
Also see section 2.4.

List of supported variables for Tixi-Bus

Type	index	access	Comment
C	0-65535	R/W	Flag (Coil)
W	0-65535	R/W	Word
DW	0-65535	R/W	DWord
F	0-65535	R/W	Float
DF	0-65535	R/W	Double
B	0-65535	R/W	Byte
S	0-65535	R/W	String (requires a “size“ entry)

4.2 ASCII protocol

The ASCII protocol is a simple option for using the FP IoT gateway to read values out of devices that provide the data as a text protocol.

The ASCII protocol is defined in the external group on the 'PROCCFG' database.

Use RS422 interface on COM2

```
<External>
   <Bus _="COM2" protocol="Tixi.Com,ASCII" type="Master"
    handshake="none" baud="115200" format="8N1">
      <Device _=”1” Pollrate=”1s”>
         <Float  _="DF"  Pos="14" End="24" acc="R" Radix="K"
          Request="&#13;" ResEnd="Ende&#13;" ResTime="10s"/>
         <Flag _="C"  Pos="0" acc="R" />
         <Word _="W"  Pos="3" End="5" acc="R" />
         <BinWord _="W"  Pos="7" Radix="B" acc="R" />
         <String _="S"  Size="30" Pos="3" Width="10" acc="R"/>
      </Device>
   </Bus>
</External>
```

Variables oft he device to be queried

The `handshake=”mode”` parameter activates various communication modes:
No entry is required for an RS232 interface.
When handshake=“HALF”, RS485 communication is used and when “FULL”, RS 422 communication is used. A special extension card is required for RS 485/422.

If the FP IoT gateway is connected to the “MB” interface (main board), ASCII communication starts directly after removing the PC cable and stops automatically when a TiXML command is detected.

A ‘Device’ entry that contains the station number and the query cycle has to be created for the device that is connected. The variable values are read in again after the query cycle elapses.

A list of variables can be defined:

Variable type in the ASCII protocol (C,W,etc.)

```
<Float  _="DF"  Pos="14" End="24" acc="R" Radix="K"
Request="&#13;" ResEnd="Ende&#13;" ResTime="10s"/>
```

Each line defines a logical name (alias, e.g. Alarm11), the variable type in the device (see: list of supported variables) and the ‘acc’ parameter contains access right ‘R’ for reading. Further parameters are described in the following sections.

The ‘def’ parameter determines the start value for the variable. A variable contains this value until the device has received the actual value from the PLC.

4.2.1 Definition of the variable query strings

The format of the variable queries can be determined for each variable using the following parameters:

Request String that is to be sent to the device to claim the data package.

Wait Specifies the time that the FP IoT gateway waits until the request string is sent to the device.

ResTime Specifies the time that the FP IoT gateway waits until the first character of a response arrives.

CharTime Specifies the maximum time between the characters. Once this time is exceeded, the response is considered complete.

If **ResTime** is used without **CharTime**, ResTime is the total time for receiving the response. After this time, the response is considered complete regardless of whether further characters arrive.

ResEnd String that marks the end of the notification. If ResEnd is not specified, only the specified timeout (ResTime/CharTime) applies. Otherwise, both must be fulfilled.

4.2.2 Evaluating the strings

The parameters apply, i.e. for all responses received after a query, until a new format for a different variable query was determined.

Find This string is searched for in the receive buffer.

Pos The data field to be read in is located in the specified position.
This refers to the start of the text that was received or, if specified, to the end of the search string (find).

FindPos This can be used to start the search from the specified position.

End The end of the data field to be read in is located in the specified position. This refers to the start of the text that was received or, if specified, to the end of the search string (find).

Width As an alternative to the end position, the data field's size can be specified.

4.2.3 Modifying the values

The data types can be modified by specifying "**Radix**".

"D" decimal
"H" hexadecimal
"O" octal
"B" binary

Floating point formats:

"K" Decimal places are separated by commas. Decimal points are evaluated as thousand separators and ignored.
An 'E/e' is evaluated as an exponent. '+/-' as sign. Anything other than numbers stops conversion.

"P" Decimal places are separated by decimal points. Commas are evaluated as thousand separators and ignored.
An 'E/e' is evaluated as an exponent. '+/-' as sign. Anything other than numbers stops conversion.

The default Radix depend on the data type:

Data type	Radix
B,W,DW,I	D
F,DF	P
C,S	“” (blank text)

Remark regarding the flags:
The flags are read in as a character. If this is "J/j/Y/y/1", this results in a TRUE (1). The characters "N/n/0" result in a FALSE (0).

List of supported variables for the ASCII protocol

Type	index	access	Comment
C	0-65535	R/W	Flag
W	0-65535	R/W	Word
DW	0-65535	R/W	DWord
F	0-65535	R/W	Float
DF	0-65535	R/W	Double
B	0-65535	R/W	Byte
S	0-65535	R/W	String
I	0-65535	R/W	Integer

4.3 Modbus RTU Master

The variables to be monitored for the Modbus controller that is connected must be defined in the FP IoT gateway.
The Modbus variables are saved in the external group on the 'PROCCFG' database:

Use COM port on COM2

```
<External>
   <Bus _="COM2" protocol="Modbus,RTU" type="Master" baud="19200">
      <Device _="1" Pollrate="1s" DWordInc="1" Timeout="3000s">
         <Coil1 _="C" ind="0x2000" acc="RW"/>
         <Input1 _="I" ind="0x03E0" acc="R"/>
         <InputReg5 _="R" ind="0x2005" acc="R"/>
         <Register10 _="H" ind="0x200A" swap="1" acc="RW" def="1"/>
         <Register3 _="D" ind="0x4003" acc="RW" def="1"/>
      </Device>
   </Bus>
</External >
```

Variables list

The BUS parameter contains the port address for the extension card, the “Modbus,RTU” protocol, the “Master” communication mode and the baud rate used.

Supported baud rates: 1200, 4800, 9600, 14400, 19200, 38400, 57600, 115200

If the FP IoT gateway is connected to the “MB” interface (main board), PLC communication starts directly after removing the PC cable and stops automatically when a TiXML command is detected.

A ‘Device’ entry that contains the station number for the connected slave PLC (‘_’ – attribute) and the query cycle has to be created for the Modbus station. The variable values are read in again after the query cycle elapses.

In the device section, special parameters that control Modbus communication between the device and the PLC are possible:

CharTimeout	Timeout between the characters (50ms,)
Pause	Pause between the notifications (50ms)
Timeout	Timeout for the response (300ms)
DwordSwap	Must be set if low before high word is sent in DWORD (0)
ForceSingleWordWrite	This should always be set to 1
UseCache	If the value is set to 0, combining subsequent variables into block queries (caching) is deactivated. All variables are collected in individual queries. (Default: 1)
MaxElements	Limits the elements queried in a Modbus notification during caching.
swap	The optional swap parameter can be used to change the sequence of the bytes within a 16 bit value for each data point. Default value=0 (MSB first; can be omitted in this case) If swap="1" is defined, the bytes are swapped within a 16 bit register (also in DWord registers) (LSB first).

A list of variables can be defined:

Variable type in the PLC (C,I,H, etc.)

```
<Alarm11 _="C" ind="0x03E3" acc="R" read="1" write="5"/>
```

Each line defines a logical name (alias, e.g. `Alarm11`) and the variable type in the Modbus controller (see: list of supported variables)
The '`ind`' parameter determines the Modbus RTU variable's address as a HEX code and the '`acc`' parameter defines the access right. The access right can either be '`R`' or '`RW`' for read or read/write access depending on the variable selected. If you add a `C` to the `acc` attribute value (e.g. '`RWC`'), the applicable variable is not queried immediately , but checked whether the following can also still be read along with it in the query (caching, if `UseCache=0`).

The '`def`' parameter determines the start value for the variable. A variable with write access contains this start value until the first write access. A variable with read access contains this value until the device has received the actual value from the PLC. If communication with the PLC exists, the start value is written to the PLC register when starting bus communication.

List of supported variables for the Modbus RTU:

Type	index	access	Comment
C	0-65535	RW	Coil (single bit)
I	0-65535	R	Discrete input
R	0-65535	R	Input register (unsigned)
H	0-65535	RW	Holding register (WORD marker, unsigned)
D	0-65534	RW	Holding register (DWORD marker, unsigned)
RI	0-65535	R	Input register (signed integer)
HI	0-65535	RW	Holding register (WORD marker, signed integer)
DI	0-65534	RW	Holding register (DWORD marker, signed integer)
RX	0-65534	R	Input register (DWORD marker, unsigned)
RY	0-65534	R	Input register (DWORD marker, signed)
DF	0-65534	RW	Holding register (DWORD) The value is interpreted as a float (only makes sense if the Modbus device really supplies a 4-byte long float value)
RY	0-65534	R	Input register (DWORD marker, signed)
RXF	0-65534	R	Input register (DWORD) The value is interpreted as a float (only makes sense if the Modbus device really supplies a 4-byte long float value)
HS	0-65534	R	Input register (WORD marker, unsigned) The contents of one or several associated input registers is interpreted as a string here. To do this, a string length with size="x" must be specified. x must be divisible by 2. Only use in conjunction with simpleType="String"! Can also be used in conjunction with the swap parameter.
D2	0-65532	RW	Holding register (QWORD = 64 bit, unsigned)
DI2	0-65532	RW	Holding register (QWORD = 64 bit, signed)
RX2	0-65532	R	Input register (QWORD = 64 bit, unsigned)
RY2	0-65532	R	Input register (QWORD = 64 bit, signed)
D2F	0-65532	RW	Holding register (QWORD = 64 bit) The value is interpreted as a float (only makes sense if the Modbus device really supplies a 8-byte long float value)
RX2F	0-65532	R	Input register (QWORD) The value is interpreted as a float (only makes sense if the Modbus device really supplies a 8-byte long float value)
RB	0-65535	R	Input register (unsigned) => interpreted as BCD (4 numbers)
HB	0-65535	RW	Holding register (WORD marker, unsigned) => interpreted as BCD (4 numbers)

Type	index	access	Comment
DB	0-65534	RW	Holding register (DWORD marker, unsigned) => interpreted as BCD (8 numbers)
RXB	0-65534	R	Input register (DWORD marker, unsigned) => interpreted as BCD (8 numbers)
D2B	0-65532	RW	Holding register (QWORD = 64 bit, unsigned) => interpreted as BCD (16 numbers)
RX2B	0-65532	R	Input register (QWORD = 64 bit, unsigned) => interpreted as BCD (16 numbers)

If only read access 'R' is possible, the 'acc' parameter can be omitted.

Modbus function codes

The FP IoT gateway uses the following Modbus function codes depending on the variable type:

Code (decimal)	Variable type
1 - Read coil status	C
2 - Read input status	I
3 - Read holding registers	H, HI, D, DI, DF, D2, DI2, HB, DB, D2B
4 - Read input registers	R, RI, RX, RY, RXF, RX2, RY2, RX2F, RB, RXB, RX2B
5 - Force single coil	C
6 – Preset single register	H, R, D, HI, RI, DI, HS, DL, DI2, D2F, HB, DB, D2B (if ForceSingleWordWrite=1)
15 – Force multiple coil	C (if size>1)

The "read" and "write" variable attributes can be used to change the function codes.

Example for reading out strings using the HS Modbus variable type:

```
<String_1 _="HS" simpleType="String" size="10" swap="1" ind="761" acc="R"/>
```

In the example, 5 subsequent holding registers (each 2 bytes long) are read out as of address 761 and interpreted as a string. Characters that cannot be printed are simply omitted.
The bytes are swapped within a 16 bit register (due to swap="1").

Remote access

The following "TransMode" command is required for remote access to the PLC: (see the TiXML reference manual for further information)

```
[<TransMode baud="19200" format="8N1" com="COM1"/>]
```

(Use `com="COM2",` if the PLC is plugged into the RS232 COM2.)

Example for 64 bits Accessing Modbus registers

As of firmware version 5.2.6.26, 64 bit wide register values can also be processed (to do this, 4x 16 bit registers are combined).
The new 64 bit variable types are called D2, DI2, RX2, RY2, D2F, RX2F
The following example illustrates using the new data types in the external:

```
[<SetConfig _="PROCCFG" ver="v">
<External>
<Bus _="ETH" Name="ModbusTCP" protocol="Modbus,TCP" type="Master">
    <Device Name="Device_1" _="1" IP="169.254.9.2" Pollrate="10s" Timeout="300ms"
     DWordInc="2" DwordSwap="0" ForceSingleWordWrite="0" UseCache="1">
```

```
        <Dbl_0 _="D2F" simpleType="Double" size="1" swap="0" ind="0" acc="RW"/>
        <Dbl_4 _="RX2F" simpleType="Double" size="1" swap="0" ind="4" acc="R"/>

        <Ui64_8 _="D2" simpleType="Uint64" size="1" swap="0" ind="8" acc="RW"/>
        <Ui64_12 _="RX2" simpleType="Uint64" size="1" swap="0" ind="12" acc="R"/>

        <I64_16 _="DI2" simpleType="Int64" size="1" swap="0" ind="16" acc="RW"/>
        <I64_20 _="RY2" simpleType="Int64" size="1" swap="0" ind="20" acc="R"/>

        <Fl_30 _="DF" simpleType="Float" size="1" swap="0" ind="30" acc="RW"/>
        <Fl_32 _="RXF" simpleType="Float" size="1" swap="0" ind="32" acc="R"/>
    </Device>
</Bus>
</External>
</SetConfig>]
```

A suitable log definition may look as follows:

```
[<SetConfig _="LOG" ver="y">
<LogDefinition>
<LogFiles>
    <JobReport size="10240"/>
    <Event size="10240"/>
    <Login size="10240"/>
    <IncomingMessage size="10240"/>
    <FailedIncomingCall size="10240"/>
    <SupportLog size="102400"/>
    <Datalogging_0 record="Datalogging_0" size="1024000" contenttype="binary"/>
</LogFiles>
<Records>
    <Datalogging_0>
        <!-- 16 Bit -->
        <H0 Name="H0" _="Uint16" path="/Process/Bus1/Device_0/H0"/>
        <H1 Name="H1" _="Int16" path="/Process/Bus1/Device_0/H1"/>

        <!-- 32 Bit -->
        <D0 Name="D0" _="Uint32" path="/Process/Bus1/Device_0/D0"/>
        <DI0 Name="DI2" _="Int32" path="/Process/Bus1/Device_0/DI0"/>
        <DF2 Name="DF2" _="Float" path="/Process/Bus1/Device_0/DF2"/>
        <RX0 Name="RX0" _="Uint32" path="/Process/Bus1/Device_0/RX0"/>
        <RY0 Name="RY0" _="Int32" path="/Process/Bus1/Device_0/RY0"/>
        <RXF2 Name="RXF2" _="Float" path="/Process/Bus1/Device_0/RXF2"/>

        <!-- 64 Bit -->
        <D20 Name="D20" _="Uint64" path="/Process/Bus1/Device_0/D20"/>
        <DI20 Name="DI20" _="Int64" path="/Process/Bus1/Device_0/DI20"/>
        <DF20 Name="D2F0" _="Double" path="/Process/Bus1/Device_0/DF20"/>
        <IRX20 Name="RX20" _="Uint64" path="/Process/Bus1/Device_0/IRX20"/>
        <IRY20 Name="RY20" _="Int64" path="/Process/Bus1/Device_0/IRY20"/>
        <IRX2F0 Name="RX2F" _="Double" path="/Process/Bus1/Device_0/IRX2F0"/>
    </Datalogging_0>
</Records>
</LogDefinition>
</SetConfig>]
```

Example for BCD Modbus variable types

As of firmware version 5.2.6.26, Modbus register values can be interpreted as BCD values.
The BCD variable types are called RB, HB, DB, RXB, D2B, RX2B.

The following example illustrates using the new BCD data types in the external:

```
[<SetConfig _="PROCCFG" ver="v">
<External>
<Bus _="ETH" Name="ModbusTCP" protocol="Modbus,TCP" type="Master">
  <Device Name="Device_1" _="1" IP="169.254.9.2" Pollrate="10s" Timeout="300ms"
   DWordInc="2" DwordSwap="0" ForceSingleWordWrite="0" UseCache="1">
      <!-- 16 bit BCD HR -->
```

```
      <Var_BCD_HB Name="Var_BCD_HB" _="HB" ind="0" acc="RW" swap="1"/>
      <!-- 16 bit BCD IR -->
      <Var_BCD_RB Name="Var_BCD_RB" _="RB" ind="0" acc="R" swap="1"/>
      <!-- 32 bit BCD HR -->
      <Var_BCD_DB Name="Var_BCD_DB" _="DB" ind="0" acc="RW" swap="1"/>
      <!-- 32 bit BCD IR -->
      <Var_BCD_RXB Name="Var_BCD_RXB" _="RXB" ind="0" acc="R" swap="1"/>
      <!-- 64 bit BCD HR -->
      <Var_BCD_D2B Name="Var_BCD_D2B" _="D2B" ind="0" acc="RW" swap="1"/>
      <!-- 64 bit BCD IR -->
      <Var_BCD_RX2B Name="Var_BCD_RX2B" _="RX2B" ind="0" acc="R" swap="1"/>
    </Device>
</Bus>
</External>
</SetConfig>]
```

A suitable log definition may look as follows:

```
[<SetConfig _="LOG" ver="y">
<LogDefinition>
<LogFiles>
    <JobReport size="10240"/>
    <Event size="10240"/>
    <Login size="10240"/>
    <IncomingMessage size="10240"/>
    <FailedIncomingCall size="10240"/>
    <SupportLog size="102400"/>
    <Datalogging_0 record="Datalogging_0" size="1024000" contenttype="binary"/>
</LogFiles>

<Records>
    <Datalogging_0>
        <!-- 16 Bit BCD -->
        <Var_BCD_HB _="Uint16" Name="16 bit BCD HR"
                    path="/Process/Bus1/Device_0/Var_BCD_HB"/>
        <Var_BCD_RB _="Uint16" Name="16 bit BCD IR"
                    path="/Process/Bus1/Device_0/Var_BCD_RB"/>
        <!-- 32 Bit BCD -->
        <Var_BCD_DB _="Uint32" Name="32 bit BCD HR"
                    path="/Process/Bus1/Device_0/Var_BCD_DB"/>
        <Var_BCD_RXB _="Uint32" Name="32 bit BCD IR"
                    path="/Process/Bus1/Device_0/Var_BCD_RXB"/>
        <!-- 64 Bit BCD -->
        <Var_BCD_D2B _="Uint64" Name="64 bit BCD HR"
                    path="/Process/Bus1/Device_0/Var_BCD_D2B"/>
        <Var_BCD_RX2B _="Uint64" Name="64 bit BCD IR"
                    path="/Process/Bus1/Device_0/Var_BCD_RX2B"/>
    </Datalogging_0>
</Records>
</LogDefinition>
</SetConfig>]
```

4.3.1 Automatic Modbus configuration

For information regarding automatic Modbus configuration, see the document Modbus_AutoKonfiguration.pdf.

4.4 Modbus ASCII Master

The way in which Modbus ASCII parameters are set it similar to the Modbus RTU (section 4.3).
The bus `protocol` parameter must be **"Modbus,ASCII"**.
The special Modbus RTU device parameters (CharTimeout, Pause, Timeout, DWordInc, DwordSwap, ForceSingleWordWrite, UseCache) are not valid for Modbus ASCII.

4.5 Modbus TCP Master

Tixi devices as of "Generation 6" (H600 and wall box series) support Modbus TCP.
The Modbus TCP variables are defined in the external section on the 'PROCCFG' database.

Use LAN interface

```
<External>
   <Bus _="ETH" protocol="ModbusTCP" type="Master" >
      <Device _="1" IP="IPAddress" Pollrate="1s" Timeout=""
       DWordInc="2" DwordSwap="0" ForceSingleWordWrite="0"
       UseCache="1">
         <Input1 _="I" ind="0x03E0" acc="R"/>
         <Coil1 _="C" ind="0x2000" acc="RW"/>
         <InputReg5 _="R" ind="0x2005" acc="R"/>
         <Register10 _="H" ind="0x200A" swap="1" acc="RW" def="1"/>
         <Register3 _="D" ind="0x4003" acc="RW" def="1"/>
      </Device>
   </Bus>
</External >
```

Set of Variables

The bus parameters define the LAN port, the "ModbusTCP" bus protocol and "Master" mode.

Device section parameters:

`IPAddress` IP address for the controller

Further parameters can be defined in the device section in order to control communication with the controller in a targeted manner:

`CharTimeout` Timeout between the characters (50ms,)

`Pause` Pause between the notifications (50ms)

`Timeout` Timeout for the response (300ms)

`DwordSwap` Must be set if low before high word is sent in DWORD (0)

`ForceSingleWordWrite`

This should always be set to 1

`UseCache` If the value is set to 0, combining subsequent variables into block queries (caching) is deactivated. All variables are collected in individual queries. (Default: 1)

`swap` The optional swap parameter can be used to change the sequence of the bytes within a 16 bit value for each data point.
Default value=0 (MSB first; can be omitted in this case)
If swap="1" is defined, the bytes are swapped within a 16 bit register (also in DWord registers) (LSB first).

Variables can be defined for each device:

Variable type in Modbus RTU protocol (e.g. C,I,R,H,D)

```
<Alarm11 _="C" ind="1" simpleType="Bit" acc="RW" def="0"/>
```

Each line defines a logical name (alias, e.g. Alarm11) and the variable type in the Modbus controller (see: list of supported variables)
The 'ind' parameter determines the Modbus TCP variable's address and the 'acc' parameter defines the access right. The access right can either be 'R' or 'RW' for read or read/write access depending on the variable selected.

The '`def`' parameter determines the start value for the variable. A variable with write access contains this start value until the first write access. A variable with read access contains this value until the device has received the actual value from the PLC. If communication with the PLC exists, the start value is written to the PLC register when starting bus communication.

List of supported variables for the Modbus TCP Master:

Type	index	access	Comment
C	0-65535	RW	Coil (single bit)
I	0-65535	R	Discrete input
R	0-65535	R	Input register (unsigned)
H	0-65535	RW	Holding register (WORD Marker, unsigned)
D	0-65534	RW	Holding Register (DWORD Marker, unsigned)
RI	0-65535	R	Input register (signed integer)
HI	0-65535	RW	Holding register (WORD Marker, signed integer)
DI	0-65534	RW	Holding Register (DWORD Marker, signed integer)
RX	0-65534	R	Input register (DWORD Marker, unsigned)
RY	0-65534	R	Input Register (DWORD Marker, signed)
DF	0-65534	RW	Holding register (DWORD) The value is interpreted as a float (only makes sense if the Modbus device really supplies a 4-byte long float value)
RY	0-65534	R	Input register (DWORD marker, signed)
RXF	0-65534	R	Input register (DWORD) The value is interpreted as a float (only makes sense if the Modbus device really supplies a 4-byte long float value)
HS	0-65534	R	Input register (WORD marker, unsigned) The contents of one or several associated input registers is interpreted as a string here. To do this, a string length with size="" must be specified. It must be divisible by 2. Only use in conjunction with simpleType="String"! Can also be used in conjunction with the swap parameter.
D2	0-65532	RW	Holding Register (QWORD = 64 Bit, unsigned)

Type	index	access	Comment
DI2	0-65532	RW	Holding Register (QWORD = 64 Bit, signed)
RX2	0-65532	R	Input Register (QWORD = 64 Bit, unsigned)
RY2	0-65532	R	Input Register (QWORD = 64 Bit, signed)
D2F	0-65532	RW	Holding register (QWORD = 64 bit) The value is interpreted as a float (only makes sense if the Modbus device really supplies a 8-byte long float value)
RX2F	0-65532	R	Input register (QWORD) The value is interpreted as a float (only makes sense if the Modbus device really supplies a 8-byte long float value)
RB	0-65535	R	Input register (unsigned) => interpreted as BCD (4 numbers)
HB	0-65535	RW	Holding register (WORD marker, unsigned) => interpreted as BCD (4 numbers)
DB	0-65534	RW	Holding register (DWORD marker, unsigned) => interpreted as BCD (8 numbers)
RXB	0-65534	R	Input register (DWORD marker, unsigned) => interpreted as BCD (8 numbers)
D2B	0-65532	RW	Holding register (QWORD = 64 bit, unsigned) => interpreted as BCD (16 numbers)
RX2B	0-65532	R	Input register (QWORD = 64 bit, unsigned) => interpreted as BCD (16 numbers)

If only read access 'R' is possible, the 'acc' parameter can be omitted.

Modbus function codes

The FP IoT gateway uses the following Modbus function codes depending on the variable type:

Code (decimal)	Variable type
1 - Read Coil Status	C
2 - Read Input Status	I
3 - Read Holding Registers	H, HI, D, DI, DF, HS
4 - Read Input Registers	R, RI, RX, RY, RXF
5 - Force Single Coil	C
6 – Preset Single Register	H, R, D, HI, RI, DI (if ForceSingleWordWrite=1)
15 – Force Multiple Coil	C (if size>1)
16 - Preset Multiple Registers	H, R, D, HI, RI, DI

The "read" and "write" variable attributes can be used to change the function codes.
Registers are written using FC16 by default. If FC6 is required, this can be activated using "ForceSingleWordWrite".

Example

Querying a Modbus TCP device at address 193.101.167.29, port 503 every second.

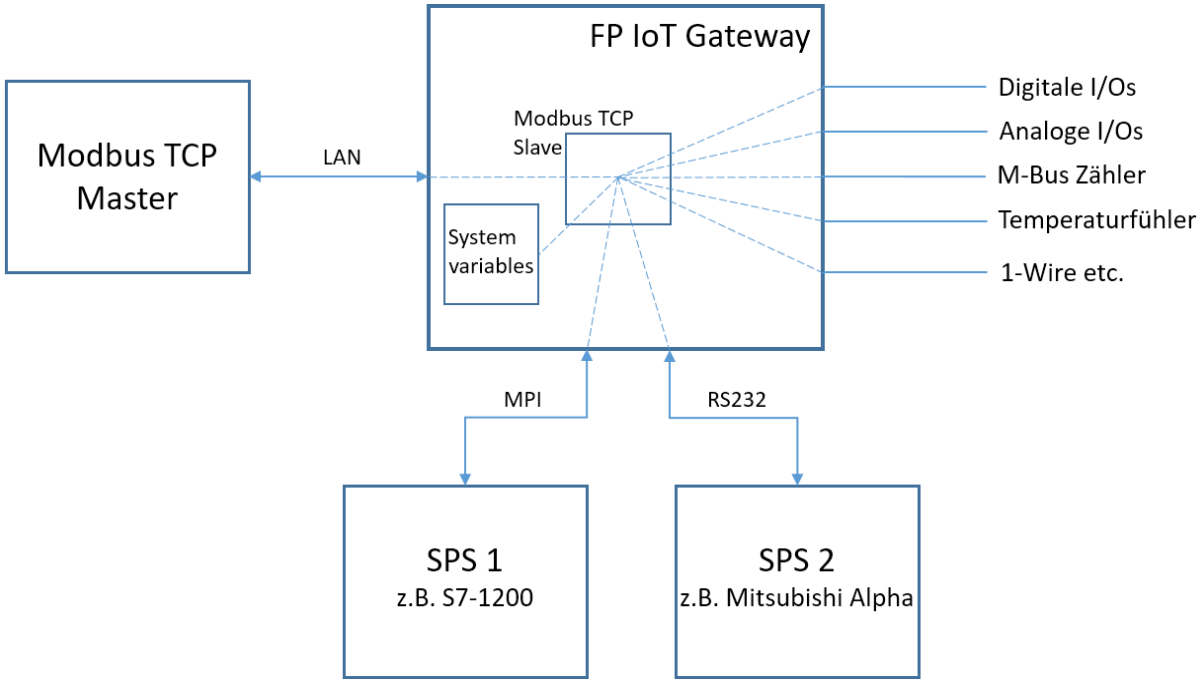
```
<Device Name="Device_31" NameUser="Software-Slave-HR" _="1"
 IP="193.101.167.29" port="503" Pollrate="1s" DWordInc="2"
 DwordSwap="0" ForceSingleWordWrite="1" UseCache="1">
   <Variable_3100 Name="H0" _="H" simpleType="Uint16" ind="0" acc="RW"/>
   <Variable_3101 Name="H1" _="H" simpleType="Uint16" ind="1" acc="RW"/>
   <Variable_3102 Name="H2" _="H" simpleType="Uint16" ind="2" acc="RW"/>
   <Variable_3103 Name="H3" _="H" simpleType="Uint16" ind="3" acc="RW"/>
   <Variable_3104 Name="H4" _="H" simpleType="Uint16" ind="4" acc="RW"/>
   <String_1 Name="String" _="HS" simpleType="String" ind="10" size="6"
             swap="1" acc="R"/>
</Device>
```

New variable types for 64 bit and BCD values

Using the new 64 bit variable types D2, DI2, RX2, RY2, D2F, RX2F and the BCD variable types RB, HB, DB, RXB, D2B, RX2B corresponds to the description in the RTU protocol (see section 4.3). These variable types can be used as of firmware version 5.2.6.26.

4.6 Modbus TCP Slave

As of firmware version 5.2.7.22, a Modbus TCP slave can be configured.
The Modbus TCP slave enables any process data defined in the FP IoT gateway to be read and written from a different control system. The data points provided by the slave are used as memory cells or there is the option to assign internal bus variables, process variables or other system properties via a reference path. The slave copies the assigned variables into its internal variable structure and provides them to the master for reading and writing purposes. For example, the Modbus TCP master can use this to read and write data points for a controller that is connected to the FP IoT gateway (e.g. Siemens, ABB or similar PLC) or a process variable defined in the FP IoT gateway.



The Modbus variables are saved in the external group on the 'PROCCFG' database:

```
<External>                                   Use LAN interface
  <Bus _="ETH" name="Bus1" protocol="ModbusTCP" type="Slave" >
    <Device _="1" Name="Device_0" Pollrate="10s">
      <Input1 _="I" ind="1000" def="0" path="/Process/PV/I1"/>
      <Coil1 _="C" ind="2000" def="1" path="/Process/Bus1/D1/C1"/>
      <InputReg5 _="R" ind="3000" def="1"/>
      <Register10 _="H" ind="4000" def="20"/>
      <Register3 _="D" ind="5000" def="41"/>
    </Device>                                          Variables list
 </Bus>
</External >
```

Device section parameters:

`Poll rate` Refresh interval for the slave variables (only when using path=)

If a reference path is used, the slave variables are refreshed at the poll rate cycle by reading and writing. At a poll rate of 10s, every 10 seconds, the values from the referenced path are transferred to the slave variable or the value written by the master in the slave variable are written to the referenced variable.

Important limitations:
1. The TCP port is set to 502 and cannot be changed.
2. Only one slave can be defined.
3. Only one master can access the slave at a time.

The slave deactivates the TCP connection after 180s inactivity.

Variable section parameters:
A list of variables can be defined:

Slave variable type (C,I,H,etc.)

```
<Coil1 _="C" ind="1000" def="0" path="/Process/Bus1/D1/C1"
simpleType="Bit" />
```

Each line defines a logical name (alias, e.g. `Coil1`) and the variable type in the Modbus slave (see table: List of supported variables for the Modbus TCP slave).

`Coil1` Variable name. For the variable types, see the “List of supported variables”.

There are 4 basic types:
C = Coil, I = Discrete input, R = Input register, H = Holding register

Further derived types are defined for basic types R and H:

Input register: R, RI, RX, RY, RX2, RY2, RXF, RX2F
Holding register: H, HI, D, DI, D2, DI2, DF, D2F

`ind` Index for access by the master.

Within the limits of the basic types specified in the table “List of supported variables for the Modbus TCP slave”, the index can be selected freely but must be unique within the basic types and must be sequential.

`def` Start value for the variable

It is essential to define a default value for each variable.
If no default value was defined, the default value is = 0.

The default values are used in the following circumstances:

- When starting the slave bus if no reference variable was used for the slave variable.
- If a reference was configured for the slave variable and the reference cannot be resolved.

`path` (optional): Reference path

A reference path can be any access path to internal data structures, e.g process variables (`/Process/PV/…`), bus variables for controllers that are connected (`BusX/DeviceY/…`), general system properties (`/GSM/…` etc.), onboard interfaces (`/Process/MB/IO/…`) or interfaces for expansion modules (`/Process/Cxxx/…`).
If a reference path is used, the slave variables are refreshed at the poll rate cycle by reading and writing, i.e. if the value for the variable specified via the reference path changes, the slave variable is also updated (for slave read variables) or the value for the variable specified via the reference path is written (for slave write variables) if the slave variable is written from the Modbus TCP master.

The user must think of a procedure for the Modbus slave to determine which registers can be called and in which order.

Please note:
The slave variables can also be written via a set command, e.g. in an EventHandler, a calculation within the ProcessVars or via the command console. If the optional path parameter is defined for a slave variable and the slave variable is written via a set command, the slave variable is updated with the reference variable's value again during the next poll cycle.

List of supported variables for the Modbus TCP slave:

Type	index	access	simpleType	Comment
C	0-65535	RW	Bit	Coil (single bit)
I	0-65535	R	Bit	Discrete input
R	0-65535	R	Uint16	Input register (unsigned)
H	0-65535	RW	Uint16	Holding register (WORD Marker, unsigned)
D	0-65535	RW	Uint32	Holding Register (DWORD Marker, unsigned)
RI	0-65535	R	Int16	Input register (signed integer)
HI	0-65535	RW	Int16	Holding register (WORD Marker, signed integer)
DI	0-65535	RW	Int32	Holding Register (DWORD Marker, signed integer)
RX	0-65535	R	Uint32	Input register (DWORD Marker, unsigned)
RY	0-65535	R	Int32	Input Register (DWORD Marker, signed)
DF	0-65535	RW	Float	Holding register (DWORD) The value is interpreted as a float
RXF	0-65535	R	Float	Input register (DWORD) The value is interpreted as a float
D2	0-65535	RW	Uint64	Holding Register (QWORD = 64 Bit, unsigned)
DI2	0-65535	RW	Int64	Holding Register (QWORD = 64 Bit, signed)
RX2	0-65535	R	Uint64	Input Register (QWORD = 64 Bit, unsigned)
RY2	0-65535	R	Int64	Input Register (QWORD = 64 Bit, signed)
D2F	0-65535	RW	Double	Holding register (QWORD = 64 bit) The value is interpreted as a double
RX2F	0-65535	R	Double	Input register (QWORD) The value is interpreted as a double

The access parameters in the table refer to the master and result implicitly from the variable type. They can therefore be omitted. For example, the master can always read and write a coil but can only read a discrete input register.

It is essential to use float as the simpleType for register types DF and RXF.
It is essential to use double as the simpleType for register types D2F and RX2F.

Modbus function codes from the Modbus master's perspective
The Modbus TCP master can use the following Modbus function codes depending on the variable type:

Code (decimal)	Variable type
1 - Read Coil Status	C
2 - Read Input Status	I
3 - Read Holding Registers	H, HI, D, DI, D2, DI2, DF, D2F
4 - Read Input Registers	R, RI, RX, RY, RX2, RY2, RXF, RX2F
5 - Force Single Coil	C
6 – Preset Single Register	H, HI, D, DI, D2, DI2, DF, D2F
15 – Force multiple coils	C (if size>1)
16 - Preset Multiple Registers	H, HI, D, DI, D2, DI2, DF, D2F

Data format for the slave variable during access via Modbus TCP

The slave variables are accessed in big-endian format.

Example for a query of a 32 bit variable with type "D", Uint32:

```
<HR2_UI32 _="D" ind="4002" def="305419896" />
```

Query 2 holding register from address `4002` (`0x0fa0`):

```
00 01 00 00 00 06 00 03 0F A2 00 02
```

Response, 4 data bytes `0x12, 0x34, 0x56, 0x78` (decimal `305419896`):

```
00 01 00 00 00 07 00 03 04 12 34 56 78
```

Rules for external configuration of the Modbus TCP slave

The following rules must be observed under all circumstances when creating the external configuration:

1. Variables of a basic type are virtually one after the other in the sequence in which they are defined in the external (there are no gaps).
2. Each index ("ind") must be unique for each basic type.
3. The indices should be created in ascending order in the external (for clarity).
   The increment can be 1 - independent of the real size of the variables.
4. During Modbus access to several registers, the start address is used to select the corresponding start variable, the requested register is filled from this and further requested registered are then filled with the subsequent variables of the same basic type as defined in the external. The indices for the following variables do not matter. The sequence in the external is decisive.
   If not enough subsequent registers exist, the system responds with an error!
   (-> Read/write more registers than defined: Error).
   When writing several variables, the target registers are described, for which complete source data exists for their variables (therefore 4 register values must be written for a 64 bit variable). The last variable is not changed in the event of an error.
5. The start variable must exist, i.e. the start address must be defined in the external as otherwise, an error message occurs.

Modbus TCP slave reference configuration

The following Modbus TCP slave reference configuration demonstrates using all available variable types. The examples with a reference path shown in the last part require the reference paths to also be available.

```
[<SetConfig _="PROCCFG" ver="y">
<External>

<!-- Modbus slave definition for all functionable register types -->
<!-- The specified access types refer to the master -->

<!-- fixer ip-port=502 -->
<Bus _="ETH" Name="Bus1" protocol="ModbusTCP" type="Slave">
  <Device _="0" Name="Device_0" Pollrate="10s">

  <!-- coil, Modbus-Access: RW, Read Coil Status(1), Force Single Coil (5),
       Force Multiple Coils (15) -->
    <!-- Coil (Bit), acc="RW", simpleType="Bit" -->
    <COIL0 _="C" ind="1000" def="0" />
```

```
  <!-- discrete input, Modbus-Access: R only, Read Input Status (2) -->
  <DIN0 _="I" ind="2000" def="1" /> <!-- Discrete Input (Bit), acc="R",
              simpleType="Bit" -->

<!-- input register, Modbus-Access: R only, Read Input Registers (4) -->

  <!-- Input Register (Word), unsigned, acc="R", simpleType="Uint16" -->
  <IR0_UI16 _="R" ind="3000" def="3000" />

  <!-- Input Register (Word), signed, acc="R", simpleType="Int16" -->
  <IR1_I16 _="RI" ind="3001" def="-3001" />

  <!-- Input Register (DWord), unsigned, acc="R", simpleType="Uint32" -->
  <IR2_UI32 _="RX" ind="3002" def="3002" />

  <!-- Input Register (DWord), signed, acc="R", simpleType="Int32" -->
  <IR3_I32 _="RY" ind="3003" def="-3003" />

  <!-- Input Register (QWord), unsigned, acc="R", simpleType="Uint64" -->
  <IR4_UI64 _="RX2" ind="3004" def="3004" />

  <!-- Input Register (QWord), signed, acc="R", simpleType="Int64" -->
  <IR5_I64 _="RY2" ind="3005" def="-3005" />

  <!-- Input Register (DWord), Interpretation als float, acc="R",
       braucht simpleType="Float" -->
  <IR10_RXF _="RXF" ind="3500" def="3.45"   simpleType="Float" />

  <!-- Input Register (QWord), Interpretation als float, acc="R",
       braucht simpleType="Double" -->
  <IR11_RX2F _="RX2F" ind="3501" def="4.56" simpleType="Double" />

<!-- holding register, Modbus-Access: RW, Read Holding Registers (3),
     Preset Single Register (6), Preset Multiple Registers (16) -->

  <!-- Holding Register (Word), unsigned,acc="RW",simpleType="Uint16" -->
  <HR0_UI16 _="H" ind="4000" def="4660" />

  <!-- Holding Register (Word), signed, acc="RW", simpleType="Int16" -->
  <HR1_I16 _="HI" ind="4001" def="-4001" />

  <!-- Holding Register (DWord), unsigned, "RW", simpleType="Uint32" -->
  <HR2_UI32 _="D" ind="4002" def="305419896" />

  <!-- Holding Register (DWord), signed, acc="RW", simpleType="Int32" -->
  <HR3_I32 _="DI" ind="4003" def="-4003" />

  <!-- Holding Register (QWord), unsigned, "RW", simpleType="Uint64" -->
  <HR4_UI64 _="D2" ind="4004" def="81985529216486895"/>

  <!-- Holding Register (QWord), signed, acc="RW", simpleType="Int64" -->
  <HR5_I64 _="DI2" ind="4005" def="-4005" />

  <!-- Holding register (DWord), interpretation as float, acc="RW",
       requires simpleType="Float" -->
  <HR10_DF _="DF" ind="4500" def="1.23"  simpleType="Float" />

  <!-- Holding register (QWord), interpretation as float, acc="RW",
       requires simpleType="Double" -->
  <HR11_D2F _="D2F" ind="4501" def="2.34"   simpleType="Double" />
```

```
    <!-- Examples with reference to an internal variable in path -->
    <!-- If reference not accessible, use default value (def) -->

    <!-- Input Register (Word), unsigned, acc="R", simpleType="Uint16" -->
    <IR6_UI16 _="R" ind="3006" def="3006"
              path="/Process/C092/AI_AAAATPPSSB/P9" />

    <!-- Input Register (Word), unsigned, acc="R", simpleType="Uint16" -->
    <IR7_UI16 _="R" ind="3007" def="999" path="/Process/PV/MyVar1" />

    <!-- Holding Register (Word), signed, "RW", simpleType="Int16" -->
    <HR20_I16 _="HI" ind="4600" def="-2000"
               path="/Process/Bus1/Device_0/HR1_I16" />

  </Device>
</Bus>
</External>
</SetConfig>]
```

4.7 M-bus

M-bus (meter bus) is a field bus protocol to monitor energy and air conditioning units in an efficient way even if several devices are connected to the FP IoT gateway.

Special M-bus devices with level converters are available for simple M-bus device connections. M-bus devices can also be connected via an external M-bus/RS232 converter (level converter).

The M-bus implementation works as a bus master with default baud rate 2400 (configurable), 8 data bits, 1 stop bit, even parity and without handshake.

A 'Device' entry must be created for each M-BUS device, which must contain at least the device's primary address (`PrimaryAddr,` decimal), secondary address (`SecondaryAddr,` 8 decimals) or fabrication address (`FabricationAddr,` 8 decimals), and the query cycle.

The M-bus variables are registered in the external group on the 'PROCCFG' database.

```
<External>
   <Bus _=”COM3” protocol=”meterbus” type="Master" baud="Baudrate"
    format="Format" Timeout="Timeout">
      <Device _=”1” PrimaryAddr=”123” SecondaryAddr="12345678"
       Pollrate=”1s”>
         <SecondaryAddr _="ident" acc="R"/>
         <Var01 ind=”1”/>
         <Var02 ind=”2”/>
      </Device>
      <Device _=”2” SecondaryAddr=”12345678” Pollrate=”60s”>
         <Var01 ind=”1”/>
         <Time _=”DateTime”/>
      </Device>
   </Bus>
</External>
```

Use M-bus interface on COM3

Device 1

Device 2

Description of the possible parameters (in italics):

baud	Baud rate. Default=2400 baud. Possible values: 300, 2400, 9600
format	Data format. Default=8E1
Timeout	Maximum waiting time between queries
PrimaryAddr	Primary address
SecondaryAddr	Secondary address (always 8 characters)
Pollrate	Query rate

If all devices on the M-bus have the same primary address, the devices can only be addressed using the secondary address. In this case, the primary address must NOT be used in the device definition!

Optional parameters are "`ManufactoryCode`" (3 ASCII characters), "`Generation`" (hex) and "`Medium`" (hex), which can be used as further differentiation characteristics for devices with the same address.

A list of variables can be defined:

```
<Var01 ind=”1”/>
```

Each line defines a logical name (alias, e.g. Var01) and the "ind" parameter defines the variable's position in the M-BUS telegram.

Outputting the device names

The “Primary address”, “Secondary address”, “Manufacturer code” and other device names can also be output when querying the device, e.g. for logging.
Special entries must be made in the variable list to do so:

```
<PrimaryAddr _="primary"/>
<SecondaryAddr _="ident"/>
<Manufacturer _="manufacturer"/>
<State _="stat"/>
<Medium _="medium"/>
<Version _="version"/>
```

(available as of FW 5.1.6.6)

Special initialisations

Time:
In order to transfer the FP IoT gateway RTC time to an M-BUS device, the “Time” variable (not readable) must be defined in the device’s device section:

```
<Time _=”DateTime”/>
```

Reset:
When communication begins, a “Reset code” can be sent to the M-BUS device. The required “ResCode” variable is not readable and contains the reset code as the “def” parameter (value=0-255):

```
<ResCode _="Reset" def="114"/>
```

Initialising raw data:
When communication begins, a user-defined datagram (def parameter) is sent to the M-BUS device:

```
<Init _="Raw" def="7304FD0834120000"/>
```

The string must be specified in hex bytes. The first byte is the CI field followed by the raw data without checksum.

VIF – Value Information Field: Medium/unit:
When reading the M-BUS variables, the device can output the VIF information regarding the medium and unit, and further values. For more information: see section 5.

Data logging:

When logging M-bus values, in order to avoid losing M-bus data, 37 bytes per value are called for (32 byte string + 5 byte data type) by default in the binary log file.
If pure M-bus numeric values are logged, the size attribute can be specified in the log file record to reduce the memory used to the actual size required.
We recommend a size of 13 bytes (8 byte value + 5 byte data type) for pure numeric values, e.g.:

```
<Datalogging_0>
 <Variable_0 _="meterbus" path="/Process/Bus1/Dev_0/Variable_0"/>
 <Variable_1 _="meterbus" size="13" path="/Process/Bus1/Dev_0/Variable_1"/>
</Datalogging_0>
```

For more information regarding data logging and calculating the log file size, see the TiXML reference manual.

Remote access

The following “TransMode” command is required for remote access to the devices: (see the TiXML reference manual for further information)

FP IoT gateway M-BUS interface:

```
[<TransMode baud="2400" format="8E1" com="COM2"/>]
```

4.7.1 M-bus scan

FP IoT gateways have a mechanism to detect all devices connected to the M-bus automatically (scan). The following prerequisites apply:

- all meters are addressed via the secondary address
- all meters use uniform settings for the baud rate and data format

The following command starts a scan on the M-bus:

```
[<ScanDevices _="COM3" protocol="meterbus" type="Master" ver="v"/>]
```

The result of a scan supplies the FP IoT gateway with a list of all meters found and their variables. Depending on the FP IoT gateway's firmware version, the scan supplies a different number of values.

Device information:

Value	Description
PrimaryAddr	Primary address
SecondaryAddr	Secondary address
Manufacturer	Manufacturer code
Version	Version
Medium	Medium
State	Status

Variable information:

Value	Description
ind	Index in M-bus telegram
vib	Value information block
value	Value
storage	storage number
tariff	Tariff information
subunit	subunit
Function	Function code

An optional reset code and individual secondary address can be specified:

```
[<ScanDevices _="COM3" protocol="meterbus" type="Master"
 SecondaryAddr="12345678" Reset="192" ver="v"/>]
```

Example 

Qundis WTT16 node (device 1) with 1 connected heat cost allocator (device 2):

```
<ScanDevices>
  <Device  PrimaryAddr="2"  SecondaryAddr="10307448"  Manufacturer="LSE"
    Version="30"  Medium="Bus/System"  State="0">
  <Var01  ind="1"  vib="On Time [h]"  value="6032"
    storage="0"  tariff="0"  subunit="0"  function="0"/>
  <Var02  ind="2"  vib="Time Point [Date+Time]"  value="2016/07/12,14:18"
    storage="0"  tariff="0"  subunit="0"  function="0"/>
  <Var03  ind="3"  vib="Model / Version"  value="3444564688926"
    storage="0"  tariff="0"  subunit="0"  function="0"/>
  <Var04  ind="4"  vib="Parameter Set Ident."  value="WTT16"
    storage="0"  tariff="0"  subunit="0"  function="0"/>
  <Var05  ind="5"  vib="Time Point [Date]"  value="19127/15/31"
    storage="0"  tariff="0"  subunit="0"  function="3"/>
  <Var06  ind="6"  vib="Bus Address"  value="258"
    storage="0"  tariff="0"  subunit="0"  function="0"/>
  <Var07  ind="7"  vib="% BATT"  value="99"
    storage="0"  tariff="0"  subunit="0"  function="0"/>
</Device>

<Device  PrimaryAddr=""  SecondaryAddr="90510717"  Manufacturer="LSE"
    Version="51"  Medium="Heat_Cost_Alloc."  State="0">
  <Var01  ind="1"  vib="Heat Cost Allocator"  value="289"
    storage="0"  tariff="0"  subunit="0"  function="0"/>
  <Var02  ind="2"  vib="Heat Cost Allocator"  value="12"
    storage="1"  tariff="0"  subunit="0"  function="0"/>
  <Var03  ind="3"  vib="Time Point [Date]"  value="2015/11/30"
    storage="1"  tariff="0"  subunit="0"  function="0"/>
  <Var04  ind="4"  vib="Time Point [Date]"  value="19127/15/31"
    storage="0"  tariff="0"  subunit="0"  function="3"/>
  <Var05  ind="5"  vib="Time Point [Date+Time]"  value="2016/07/12,10:36"
    storage="0"  tariff="0"  subunit="0"  function="0"/>
  <Var06  ind="6"  vib="Size of Storage"  value="8"
    storage="8"  tariff="0"  subunit="0"  function="0"/>
  <Var07  ind="7"  vib="Time Point [Date]"  value="2016/06/30"
    storage="15"  tariff="0"  subunit="0"  function="0"/>
  <Var08  ind="8"  vib="Storage Interval [months]"  value="1"
    storage="8"  tariff="0"  subunit="0"  function="0"/>
  <Var09  ind="9"  vib="Heat Cost Allocator"  value="289"
    storage="15"  tariff="0"  subunit="0"  function="0"/>
  <Var10  ind="10"  vib="Heat Cost Allocator"  value="289"
    storage="14"  tariff="0"  subunit="0"  function="0"/>
  <Var11  ind="11"  vib="Heat Cost Allocator"  value="276"
    storage="13"  tariff="0"  subunit="0"  function="0"/>
  <Var12  ind="12"  vib="Heat Cost Allocator"  value="259"
    storage="12"  tariff="0"  subunit="0"  function="0"/>
  <Var13  ind="13"  vib="Heat Cost Allocator"  value="226"
    storage="11"  tariff="0"  subunit="0"  function="0"/>
  <Var14  ind="14"  vib="Heat Cost Allocator"  value="207"
    storage="10"  tariff="0"  subunit="0"  function="0"/>
  <Var15  ind="15"  vib="Heat Cost Allocator"  value="63"
    storage="9"  tariff="0"  subunit="0"  function="0"/>
  <Var16  ind="16"  vib="Heat Cost Allocator"  value="12"
    storage="8"  tariff="0"  subunit="0"  function="0"/>
  <Var17  ind="17"  vib="Model / Version"  value="2645700378675"
    storage="0"  tariff="0"  subunit="0"  function="0"/>
  <Var18  ind="18"  vib="Bus Address"  value="258"
    storage="0"  tariff="0"  subunit="0"  function="0"/>
</Device>
```

Note:
The scan can sometimes take a very long time if many M-bus devices are connected to the bus (up to several hours).

4.7.2 Starting to read out the M-bus

The M-bus meters that are connected can be read out in a targeted manner using the following command:

```
[<StartBusPolling _=“COM-Port“/>]
```

Busname = `COM3 [| COM1 | COM2]`

As soon as the command was started, you can see this in the corresponding system property:

`/Process/COM1PollActive 0` = not active `1` = polling running
`/Process/COM2PollActive 0` = not active `1` = polling running
`/Process/COM3PollActive 0` = not active `1` = polling running

Example 

Starting M-bus polling on COM3:

```
[<StartBusPolling _=”COM3”/>]
```

The following system property is created:

`/Process/COM3MBusPollActive` `0` = not active `1` = MBus polling running

4.8 CAN bus

Not yet released.

4.9 CS protocol (EN 62056-21 Mode C / EN 61107)

FP IoT gateways support the CS protocol in accordance with EN 62056-21 mode C (only data query). This protocol is often used in electricity meters.

The variables are registered in the external group on the 'PROCCFG' database.

Use COM2 interface

```
<External>
   <Bus _="COM2" Name="BusName" protocol="DIN1107,CSData"
    type="Master" baud="Speed" handshake="HALF" format="DataFormat"
    BusPollrate="BusPollrate TUnit">
      <Device _="1" Address="xxxxxx" Pollrate="1s" PreSend="PreSend"
       PreDelay="PreDelay" OpMode="OpMode" Options="Options">
         <Manufacturer _="MC" acc="R" simpleType="String"/>
         <Identification _="ID" acc="R" size="16"
          simpleType="String" format=";%% "/>
         <DevID_OBISCode _="OBIS" obis="0.0.0" acc="R" size="5"
          simpleType="String" format=";%% "/>
         <DevID_DevID _="Text" obis="0.0.0" acc="R" size="8"
          simpleType="String" format=";%% "/>
         <DevID_DevIDDW _="DWord" obis="0.0.0" acc="R"
          simpleType="Uint32" format=";%% "/>
         <Cnt_OBISCode _="OBIS" obis="3.8.1" acc="R" size="5"
          simpleType="String" format=";%% "/>
         <Cnt_CounterDbl _="Count" obis="3.8.1" acc="R"
          simpleType="Double" format="F09,3;%% "/>
         <Cnt_Counter _="DWord" obis="3.8.1" acc="R"
          simpleType="Uint32" format="F09,3;%% "/>
         <Cnt_Unit _="Text" obis="3.8.1" acc="R" size="3"
          subind="1" simpleType="String" format=";%% "/>
         <Cnt_Date _="Text" obis="1.6.1" acc="R" size="8"
          simpleType="String" format=";(%%)"/>
         <OBIS_DevIDCode _="OBISLine" ind="1" acc="R" size="15"
          simpleType="String" format=";%% "/>
         <OBIS_Error _="OBISLine" ind="0" acc="R" size="15"
          simpleType="String" format=";%% "/>
         <Obis_180 _="Count" ProgCmd="R1" simpleType="Double"
           obis="1.8.0" subindVal="0" acc="R"/>
      </Device>
   </Bus>
</External>
```

The meters are connected to an RS485 interface on the FP IoT gateway.
All XML attributes shown in red are required attributes with constant values.

4.9.1 Bus configuration

Description of the constant attributes (marked in red):

protocol	Defines the bus protocol: "DIN1107,CSData"
type	Defines the FP IoT gateway's role in the bus as "Master"
handshake	Determines the type of flow control between the FP IoT gateway and the meter. When using the 2-wire RS485 bus, a half-duplex connection must be configured, therefore "HALF"
COM	required FO IoT gateway interface to which the bus is connected. If 2 buses are defined, the interfaces must differ.

Value	Description
COM2	RS 485 interface on COM2
COM*X*	Further RS485 interface (if present in the FP IoT gateway) *X* = interface number

Overview of possible parameters (values written in italics):

BusName optional, the default is: "" (empty). Maximum of 20 alpha-numeric characters, must not begin with a number.

Defines the name of the TiXML attribute group that represents the bus in the "Process" attribute group. The group name is the first part of a device variable address:

Process/***Bus***/*Device/Variable*

The names written in italics are defined by the process configuration. The name written in bold is defined by this attribute.

Value	Description
"" (empty)	(Default value) auto name. The attribute group is assigned an automatic name: Process/Busn/... n automatically-generated index number
BusName (not empty)	The attribute group receives the specified name: Process/MyName/... Example: <Bus Name="CS" ...> Process/CS/...

Speed optional, default: 9600
Valid values: 300, 600, 1200, 2400, 4800, **9600**, 19200, 38400, 115200

DataFormat optional, default: 7E1
Determines the data formal on the serial interface.
Syntax: **DataBitsParityBitsStopBits**

DataBits: 8 = 8 data bits; 7 = 7 data bits
ParityBits: N = no parity bit; E = even parity; O = odd parity
StopBits: 1 = 1 stop bit; 2 = 2 stop bits

BusPollrate optional, default: "200ms"

Time in time units (see TUnit), for which the FP IoT gateway waits between two bus queries at least. This prevents the processor becoming blocked in the event of an "empty" bus configuration.

Value range:
0ms - 1073741823ms
0s - 1073740s
0m - 17894m
0h - 297h

TUnit Time unit (ms, s, m, h)

4.9.2 Device configuration

The device configuration is defined by a “Device” XML element within the bus element and described by the following XML attributes:

```
<External>
   <Bus ...>
      <Device _="ID" Name="Alias" Address="Address" PollRate="Rate
       TUnit" Opmode="OpMode" Options="Options">
              <<< Configuration of the device variables >>>
      </Device>
   </Bus>
</External>
```

A “Device” entry must be configured for each meter that is connected to the interface (COM2 or COMX) and that is to communicate with the FP IoT gateway.

Description of the *(italic and bold)* attribute values*:*

ID, required, numbers 0 - 255

Meter bus ID. The address is not required in principle, as no sequential numbers are required for the device in the CS protocol.
It is used for differentiation if several devices are configured.

Address optional, default:““(empty)
String of alphanumeric characters and spaces, max. 30 characters

Meter communication address (generally the serial number).
The address is only required if several meters are connected to a bus. If the address is empty (default), each device is addressed (see EN 62056-21:2002).

Alias optional, default value: “” (empty), maximum of 20 alpha-numeric characters, must not begin with a number.

Defines the name of the TiXML attribute group that represents the device in the "Bus” attribute group. The group name is the second part of a device variable address:

Process/*Bus*/***Device***/*Variable*

The names written in italics are defined by the process configuration. The name written in bold is defined by this attribute.

Value	Description
“” (empty)	(Default value) auto name. The attribute group is assigned an automatically-generated name: Process/*BusName*/D*Id…* *Id* Value of the “_” attribute
DeviceName (not empty)	The attribute group receives the specified name: Process/*Bus/DeviceName…* Example: <Bus Name=”CS” ….> <Device Name=”Meter1” ….> Process/CS/Meter1/…

Rate optional, default: “15m”, for valid values, see the poll rate in time units below (see TUnit), with which the FP IoT gateway queries the device.
For value ranges, see "BusPollrate"

TUnit Time unit (ms, s, m, h)

PreSend sends the specified value (0-255) binary before starting communication with the meter

PreDelay Waiting time in ms after sending the value specified in PreSend

OpMode Operating mode; default: “data” (can then be omitted)
“prog” = programming mode. This mode is required for special commands such as “R1”.

Options Options (optional; default = no options)
“RemainConnected” = In this mode, the connection to the meter is retained during each cycle. The meter values can therefore be queried quicker. This mode is often used in conjunction with “prog” OpMode and R1 commands.
Attention: Only one single meter may be connected within the bus in “RemainConnected” mode!

4.9.3 Device variable configuration

The configuration for a device variable is defined by an XML element within the “Device” element to which the device variable belongs. The following XML attributes determine the configuration for a variable:

```
<External>
  <Bus …>
     <Device …>
        <ValueName  _="MC"  format="Format"/>
        <ValueName  _="ID" sizeID="IDSize" format="Format"/>
        <ValueName  _="OBIS" obis="OBIS" size="Size" indVAL="OBISInd"
          format="Format" />
        <ValueName  _="Text" obis="OBIS" size="Size" indVAL="ValInd"
          subindTXT="TxtSubInd" format="Format" />
        <ValueName  _="Count" obis="OBIS" precision="Precision"
         indVAL="ValInd" subindVAL="ValSubInd" format="Format" />
        <ValueName  _="DWord" obis="OBIS” indVAL=" ValInd "
         subindVAL="ValSubInd" exp="Exp" multip="Factor" format="Format" />
        <ValueName  _="Byte" obis="OBIS” indVAL=" ValInd "
        subindVAL="ValSubInd " exp="Exp" multip="Factor" format="Format" />
        <ValueName  _="OBISLine"  indVAL="LineInd" size="Size"
         format="Format" />
        <ValueName _="Count" ProgCmd="R1" simpleType="Double"
         obis="OBIS" subindVal="0" />
     </Device>
  </Bus>
</External>
```

Each element describes a device parameter value that can be read by the meter in order to generate alarm messages or log the device status for example.

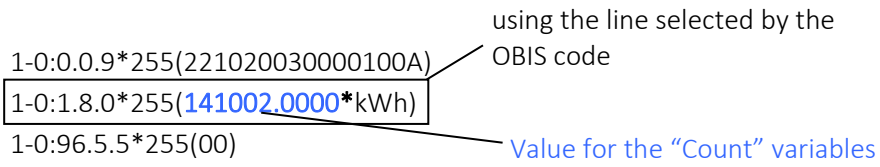
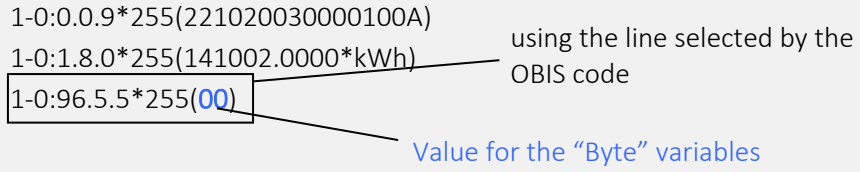
There are eight classes of device variables. One class combines all device parameters that have the same properties (storage, encoding, reading or writing possible).
The value for the type attribute "_" (highlighted in red) defines the class to which a device variable belongs:

Variable class	Description

Text	Parameter value represented by an ASCII text string in OBIS format.
	The value for a "Text" device variable is saved in the FP IoT gateway as a C-string (simpleType="**String**"). The "Text" parameters have a maximum length that is defined by the "size" attribute.
	The "**obis**" attribute indicates a line in OBIS format from which the value is read.
	The "**indVAL**" attribute defines which value (bracket) is to be read from the line.
	If a value comprises several parts, e.g. meter value and unit (separated by * characters), the "subind" attribute is used to address the corresponding part of the value. If the "**subindTXT**" value is 255 (default value), the entire bracket is read without paying attention to the value parts.
	Example
	Reading the device identification
	`<DevID_DevID _="Text" obis="1-0:0.0.9*255" indVAL="0" subindTXT="0" size="16" />`
	
	User data sent by the meter:
	1-0:0.0.9*255(2210200300001000A)
	1-0:1.8.0*255(141002.0000*kWh)
	1-0:96.5.5*255(00)
	using the line selected by the OBIS code
	Value for the "Text" variables
	
	Example
	Reading the meter value unit
	`<Cnt_Unit _="Text" obis="1-0:1.8.0*255" size="3" indVAL="0" subindTXT="1"/>`
	
	User data sent by the meter:
	1-0:0.0.9*255(221020030000100A)
	1-0:1.8.0*255(141002.0000*kWh)
	1-0:96.5.5*255(00)
	using the line selected by the OBIS code
	Value for the "Text" variables
	
	Example
	Reading the entire meter value with unit as text
	`<Cnt_Unit _="Text" obis="1-0:1.8.0*255" size="3" indVAL="0" subindVAL="255" />`

Variable class	Description
	User data sent by the meter: 1-0:0.0.9*255(221020030000100A) 1-0:1.8.0*255(141002.0000*kWh) 1-0:96.5.5*255(00) using the line selected by the OBIS code Value for the "Text" variables *Example* Reading the meter value time stamp as text `<Cnt_DateRAW _="Text" obis="1-0:1.8.0*255" size="10" indVAL="1" subind="255"/>` User data sent by the meter: 1-0:0.0.9*255(221020030000100A) 1-0:1.8.0*255(141002.0000*kWh)(0504010000) 1-0:96.5.5*255(00) using the line selected by the OBIS code Value for the "Text" variables
OBIS	OBIS code represented by an ASCII text string in OBIS format. The value for an "OBIS" device variable is saved in the FP IoT gateway as a C-string (simpleType="**String**"). The "OBIS" parameters have a maximum length that is defined by the "size" attribute. The "**obis**" attribute indicates a line in OBIS format from which the value is read. The "**ind**" attribute defines which part of the OBIS code is to be read (the default is 0 = complete OBIS code). *Example* Reading the entire OBIS code `<DevID_OBIS _="OBIS" obis="1-0:0.0.9*255" indVAL="0" size="16" />` User data sent by the meter: 1-0:0.0.9*255(221020030000100A) 1-0:1.8.0*255(141002.0000*kWh) 1-0:96.5.5*255(00) using the line selected by the OBIS code Value for the "OBIS" variables *Example* Reading the (optional) first part (medium)

Variable class	Description
	<Cnt_Medium _="OBIS" obis="1-0:1.8.0*255" size="3" indVAL="1"/>** User data sent by the meter: 1-0:0.0.9*255(221020030000100A) 1-0:1.8.0*255(141002.0000*kWh) 1-0:96.5.5*255(00) using the line selected by the OBIS code Value for the "OBIS" variables *Example* **Reading the middle (obligatory) part** **<Cnt_OBISID** _="OBIS" obis="1-0:1.8.0*255" size="5" indVAL="2" />** User data sent by the meter: 1-0:0.0.9*255(221020030000100A) 1-0:1.8.0*255(141002.0000*kWh) 1-0:96.5.5*255(00) using the line selected by the OBIS code Value for the "OBIS" variables *Example* **Reading the (optional) third part (stored value)** **<Cnt_VV** _="OBIS" obis="1-0:1.8.0*255" size="3" indVAL="3" />** User data sent by the meter: 1-0:0.0.9*255(221020030000100A) 1-0:1.8.0*255(141002.0000*kWh)(0504010000) 1-0:96.5.5*255(00) using the line selected by the OBIS code Value for the "OBIS" variables
Count	Meter value represented by a fixed-point number in OBIS format. The value for a "Count" device variable is saved in the FP IoT gateway as a double precision floating point number (simpleType="**Double**"). **Note:** Count variables are intended for saving meter readings. The meter readings contain up to 10 decimal places. This type of number cannot be saved in a DWORD integer value (maximum of 9 decimal places). Double values provide the required precision (maximum of 16 decimal places). As only one integer arithmetic is available for further calculations in the FP IoT gateway, the "**precision**" attribute must be used to convert the double value into an integer value. Precision losses may occur.

Variable class	Description
	The "**obis**" attribute indicates a line in OBIS format from which the value is read. The "**indVAL**" attribute defines which value (bracket) is to be read from the line (the default is 0). If a value comprises several parts, e.g. meter value and unit, the "**subindVAL**" attribute is used to address the corresponding part of the value. The default value is 0 so that the meter value is always read if "ind" and "subind" are not specified. *Example* **Reading the meter value.** `<Cnt_Counter _="Count" obis="1-0:1.8.0*255" indVAL="0" subindVAL="0" />` User data sent by the meter: 
Byte	Parameter value represented by an ASCII-encoded integer number (maximum of 3 numbers) in OBIS format. The value for a "Byte" device variable is saved in the FP IoT gateway as a byte number (simpleType="**Uint8**"). The "**obis**" attribute indicates a line in OBIS format from which the value is read. The "**indVAL**" attribute defines which value (bracket) is to be read from the line (the default is 0). If a value comprises several parts, e.g. meter value and unit, the "**subindVAL**" attribute is used to address the corresponding part of the value. The default value is 0 so that the numeric value is always read if "indVAL" and "subindVAL" are not specified. *Example* **Reading the status value as a byte value.** `<State_State _="Byte" obis="1-0:96.5.5*255" />` User data sent by the meter: 
DWord	Parameter value represented by an ASCII-encoded integer number (maximum of 10 numbers) in OBIS format. The value for a "Byte" device variable is **saved** in the FP IoT gateway as a double word number (simpleType="**Uint32**", see section 2.1).

Variable class	Description
	The "**obis**" attribute indicates a line in OBIS format from which the value is read. The "**indVAL**" attribute defines which value (bracket) is to be read from the line (the default is 0). If a value comprises several parts, e.g. meter value and unit, the "**subindVAL**" attribute is used to address the corresponding part of the value. The default value is 0 so that the numeric value is always read if "indVAL" and "subindVAL" are not specified. *Example* **Reading the meter value time stamp as a numeric value** `<Cnt_Date _="Text" obis="1-0:1.8.0*255" indVAL="1" />` User data sent by the meter: using the line selected by the OBIS code 1-0:0.0.9*255(221020030000100A) 1-0:1.8.0*255(141002.0000*kWh)(0504010000) 1-0:96.5.5*255(00) Value for the "DWord" variables
OBISLine	Saves one line of the OBIS format as ASCII text. This variable class should enable the OBIS text that is sent from the meter to be forwarded in its original form to an accounting system. The "OBISLine" parameters have a maximum length that is defined by the "size" attribute. The "indVAL" attribute defines the number of the line that is to be saved. *Example* **Reading the OBIS line with the meter value** `<OBIS_Cnt _="OBISLine" size="40" indVAL="1" />` User data sent by the meter: using the line selected by the "ind" attribute 1-0:0.0.9*255(221020030000100A) 1-0:1.8.0*255(141002.0000*kWh) 1-0:96.5.5*255(00) Value for the "OBISLine" variables
MC	**Manufacturer code** represented by three ASCII characters in the CS protocol. The value for an "MC" device variable is saved in the FP IoT gateway as a C-string (simpleType="**String**"). The string length is fixed at **3 characters**. The value for an "MC" device variable is determined from the CS protocol telegram in the following way: Start character Baud rate Separation End /XXX5NNNNNNNNNNNNNNNN*CRLFCRLF(user data)*!*CRLF* Value for the "MC" variables (3 ASCII characters) Identification (max. 16 ASCII characters)

Variable class	Description
ID	**Identification** represented by a maximum of 16 ASCII characters in the CS protocol. The value for an "ID" device variable is saved in the FP IoT gateway as a C-string (simpleType="**String**", see section 2.1). The "ID" parameters have a maximum length that is defined by the "size" attribute. The value for an "ID" device variable is determined from the CS protocol telegram in the following way: Start character Baud rate Separation End **/XXX5**NNNNNNNNNNNNNNNN*CRLFCRLF(user data)!CRLF* Value for the "MC" variables (3 ASCII characters) Identification (max. 16 ASCII characters)
R1	Special query command to read out individual values using OBIS code. Accelerates the data query significantly when reading out only a few values. OpMode="prog" is required for this.

Each class has its own set of attributes.

Description of the (italic and bold) attribute values:

ValueName Device variable name, maximum of 30 characters, only alphanumeric characters, no umlauts, must not begin with a number. Forms the last part of the device variable address:

Process/*Bus*/*Device*/***VariableName***

The names written in italics are defined by the process configuration. The name written in bold is defined by this value.

Example

```
<Bus Name="CS" … >
   <Device Name="Meter1" … >
      <Cnt_Counter…../>
```

Medium A	–	Kanal B	:	Messgröße C	;	Messart D	;	Tarif E	*	Vorwert F	Daten

Process/ CS /Meter1/Cnt_Counter

OBISInd optional, default value: 0 decimal number (0 - 255)
Indicates the **index for an OBIS code part** in an OBIS format line.

The code has the following layout:

Medium A	–	Kanal B	:	Messgröße C	;	Messart D	;	Tarif E	*	Vorwert F	Daten

Fields A+B and F are optional.
The code therefore comprises three parts that can be addressed using this attribute.
The default value 0 supplies the complete (received) code.

Example

Individual parts

1-0:1.8.0*255(141002.0000*kWh)(0504010000)

Value ind="2"

Value ind="1"

Counting starts with the first part for 1.

Entire code

1-0:1.8.0*255(141002.0000*kWh)(0504010000)

Value ind="0"

ValInd optional, default value: 0 decimal number (0 - 255)

Indicates the **index for a value** (bracket) in an OBIS format line.

Example

1-0:1.8.0*255(141002.0000*kWh)(0504010000)

OBIS code

Value ind="0"

Value ind="1"

Counting starts with the first bracket for 0.

ValSubInd optional, default value: 0 decimal number (0 - 255)

Indicates the **index for a value part** (within a bracket) in an OBIS format line. A value can comprise several parts that are separated by the "*" character or by spaces. These parts can be addressed individually using this index.

Example

Value subind="0"

1-0:1.8.0*255(141002.0000*kWh)(0504010000)

OBIS code

Value subind="0"

Value subind="1"

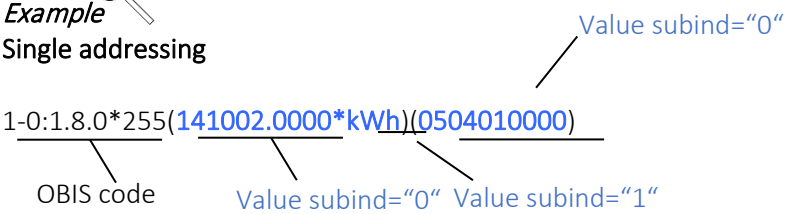
Counting starts with the first part of a value (bracket) for 0.

TxtSubInd optional, default value: 255 decimal number (0 - 255)

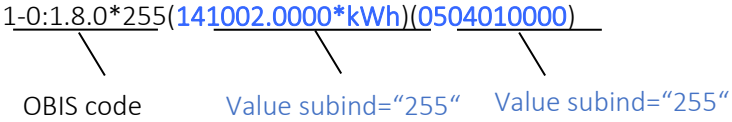
Indicates the **index for a value part** (within a bracket) in an OBIS format line or the **entire bracket**. A value can comprise several parts that are separated by the "*" character or by spaces. These parts can be addressed individually using this index or combined using **index 255**.

Example

Single addressing



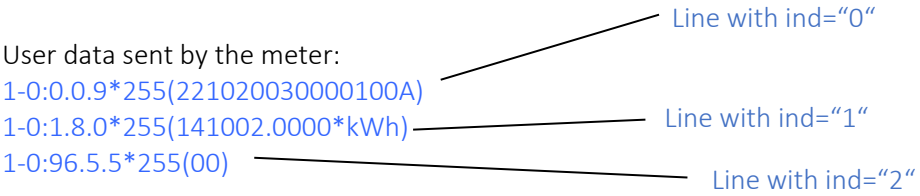
Complete addressing



Counting starts with the first part of a value (bracket) for 0.

LineInd optional, default value: 0, numeric value 0 - 255
Meaning only for variables in the **OBISLine** class. Describes the line's index in the user data for the CS protocol.

Example



Counting starts with the first line in the user data for 0.

Size required, 0 - 100
Meaning only for variables in the **Text** class. Describes the maximum number of ASCII characters in OBIS format.

IDSize optional, the default value is "16", 1 - 16
Meaning only for variables in the **ID** class. Describes the maximum number of ASCII characters in the CS protocol.

OBIS required, OBIS code search screen
OBIS code to address the line in OBIS format.

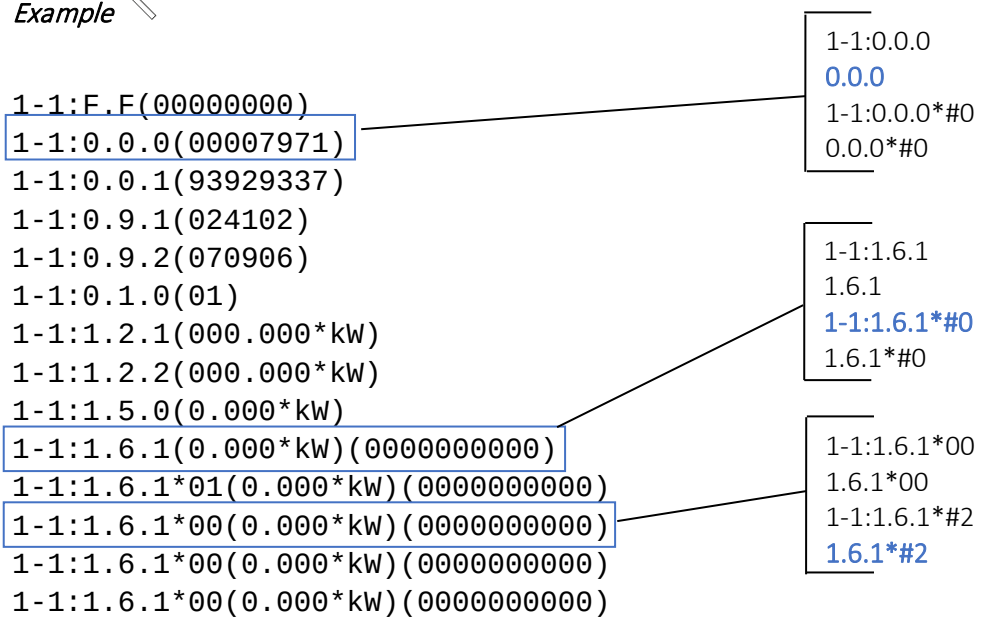
The code has the following layout:

Medium A	–	Kanal B	:	Messgröße C	.	Messart D	.	Tarif E	*	Vorwert F	Daten

Fields A+B and F are optional.
In order to determine the referenced row, the code sent by the meter is compared to the code specified in the "obis" attribute. Both codes are the same if fields C,D and E match. The selection can be made more precise by specifying further fields.
If no stored value (section F) is specified, the first value that correlates to the remaining specifications is found.

With **#0** as the stored value, the first value that does not have a stored value but where the remaining specifications correlate is found.
With ***#n,*** where ***n*** is a number between 1 and 255, the value with the nth stored value is found.

Example



Left: Meter data output. The lines that were found receive a frame.
Right: OBIS code search screen. The best search screens are in bold.

Format optional, the default value is "" (no formatting).
The format options designate the default formatting for the value as it is output, e.g. in e-mails or in the Get command (see the "TiXML Reference Manual").

Factor optional, default value: 1/1, format: see below
The value received from the meter (valueDevice) is multiplied by this factor:

valueTDG = Factor * valueDevice
valueDevice = 1/Factor * valueTDG

The factor is written as a break, e.g.: "1/1000" or "3600/1", the meter and the denominator must be positive whole numbers and must not be 0.

Exp optional, default value: 0, permitted values: see the table below
Exponent of basis 10 that defines the precision of a fixed-point number.
The value saved in the FP IoT gateway is multiplied by 10 exp(*Exp*) (after applying the factor) to form the variable value.

valueVariable = 10^{Exp} * valueGateway.

The exponent therefore defines the position of the decimal point in the fixed-point number.

The following values are possible:

Exp value	Description
-6	Precision = 0.000001
-5	Precision = 0.00001
-4	Precision = 0.0001
-3	Precision = 0.001
-2	Precision = 0.01
-1	Precision = 0.1
0	Precision = 1 (default)
1	Precision = 10
2	Precision = 100
3	Precision = 1000
4	Precision = 10000
5	Precision = 100000
6	Precision = 1000000

Precision optional, default value:0, valid values: see the following table

The value saved in the FP IoT gateway is multiplied by 10 exp(*Exp*) in order to convert the value to integer representation. Integer representation is used when calculating the process variables, e.g using the commands (GT, LT etc.). It therefore specifies the precision when calculating the process variables and must be defined depending on the application.

Integer representation = whole number(10^{Exp} * valueParameter).
The values correspond to the following table.

Precision Value	Description
-6	Factor = 0.000001
-5	Factor = 0.00001
-4	Factor = 0.0001
-3	Factor = 0.001
-2	Factor = 0.01
-1	Factor = 0.1
0	Factor = 1 (default)
1	Factor = 10
2	Factor = 100
3	Factor = 1000
4	Factor = 10000
5	Factor = 100000
6	Factor = 1000000

4.9.4 Error value for a variable

Each configured device variable has a corresponding error state value.
The error state value contains a two-tier error code that describes the exact error state and the error source. The error state value can be used to discover configuration errors or to generate alarms or log entry if a communication error occurs.

If a variable cannot be read because communication or application errors occur, the error state value is provided with the corresponding error code and the variable value set to "undefined".

In order to read the error state value for a variable, the expansion of the "Get" command is used:

Command:
```
[<Get  _="VPath"  AddInfo="AddInfo" />]
```

Overview of possible parameters (values written in italics):

Vpath: Path to address the parameter.

AddInfo:
Error ..... Returns the value's error state.

AddInfo is ignored for values that are not parameters for external devices.
The parameter's value is returned.

Response:
```
[<Get _="ErrorClass,ErrorValue" />]
```

ErrorClass: Error class
0 No error
1 Error detected by the FP IoT gateway

ErrorValue: Error value

ErrorClass	ErrorValue	Meaning
0	0	No error
1	2	The value line is not contained in the telegram. This occurs if: a) No line with the OBIS code defined in the configuration for a device variable (see the "obis" attribute) was found. or b) If no line with the line index defined in the configuration for a device variable (see the "ind" attribute for the "OBISLine" variable class) was received
1	3	The value is not contained in the telegram. This occurs if a line was found but the value addressed by the "ind" and "subind" attribute (see device variable configuration) was not found or could not be converted.
1	4	Invalid telegram. A telegram that does not have the expected structure or that has an incorrect block control code was found.
1	5	Timeout, no complete telegram received after the last query.

Example for using the R1 command

```
<Device Name="Device_0" NameUser="MT174" _="1" OpMode="prog"
  Options="RemainConnected" Pollrate="1s" Address="12345678"
  PreSend="0" PreDelay="180">

    <Variable_0 Name="Uhrzeit""Time" _="Text" size="6" ProgCmd="R1"
     obis="0.9.1" acc="R"/>

    <Obis_000 Name="Nummer""Number" _="Text" ProgCmd="R1"
simpleType="String"
     obis="0.0.0" size="8" acc="R"/>

    <Obis_092 Name="Datum""Date" _="Text" ProgCmd="R1" simpleType="Uint32"
    obis="0.9.2" size="12" subindVAL="0" acc="R"/>

    <Obis_091 Name="Uhrzeit""Time" _="DWord" ProgCmd="R1"
    simpleType="Uint32" obis="0.9.1" subindVAL="0" acc="R"/>

    <Obis_170 _="Count" ProgCmd="R1" simpleType="Double" obis="1.7.0"
    subindVAL="0" acc="R"/>

    <Obis_180 _="Count" ProgCmd="R1" simpleType="Double" obis="1.8.0"
    subindVAL="0" acc="R"/>

    <Obis_280 _="Count" ProgCmd="R1" simpleType="Double" obis="2.8.0"
    subindVAL="0" acc="R"/>

    <Obis_270 _="Count" ProgCmd="R1" simpleType="Double" obis="2.7.0"
    subindVAL="0" acc="R"/>

</Device>
```

4.10 D0 protocol (EN 62056-21 Mode D)

FP IoT gateways support the CS protocol in accordance with EN 62056-21 mode D.

This protocol is often used in electricity meters (electronic domestic supply meter).

The bus definition and the variable definition are virtually identical to the CS protocol (see section 4.9), therefore, only the differences are documented here.

External group on the 'PROCCFG' database:

```
<External>
   <Bus _="COM2" Name="BusName" protocol="DIN1107,D0" type="Master"
    baud="Speed" handshake="HALF" format="DataFormat"
    BusPollrate="BusPollrate TUnit">
```

Differences

- Protocol "DIN1107,D0"
- Only one meter can be connected to each serial interface (no bus!)

4.11 SML protocol

8th generation FP IoT gateways (e.g. HE852) support the SML protocol for the LMN interface for modern measuring equipment. The LMN interface works with a baud rate of 921600 bps.
The variables are registered in the external group on the 'PROCCFG' database.

Use COM2 interface

```
<External>
   <Bus _="COM2" Name="BusName" product="LMN" family="LMN"
    protocol="LMNProt" type="Master" baud="Speed" handshake="HALF"
    format="DataFormat">
      <Device _="1" Name="Alias " MS-ID="MS_ID" Pollrate="1s">
         <Obis_180 _="Count" obis="1.8.0" simpleType="Double" acc="R" />
         <Obis_280 _="Count" obis="2.8.0" simpleType="Double" acc="R" />
         <Obis_3270 _="Count" obis="32.7.0" simpleType="Double" acc="R" />
      </Device>
   </Bus>
</External>
```

The meters are connected to an RS485 interface on the Tixi G8 device.
All XML attributes shown in red are required attributes with constant values.

4.11.1 Bus configuration

Description of the constant attributes (marked in red):

protocol Defines the bus protocol: "LMNProt"

type Defines the FP IoT gateway's role in the bus as "Master"

handshake Determines the type of flow control between the FP IoT gateway and the meter. When using the 2-wire RS485 bus, a half-duplex connection must be configured, therefore "HALF"

COM required
FO IoT gateway interface to which the bus is connected.
If 2 buses are defined, the interfaces must differ.

Value	Description
COM1	RS 485 interface on COM1
COM*X*	Further RS485 interface (if present in the FP IoT gateway) *X* = interface number, e.g. 2

Overview of possible parameters (values written in italics):

BusName optional, the default is: "" (empty). Maximum of 20 alpha-numeric characters, must not begin with a number.

Defines the name of the TiXML attribute group that represents the bus in the "Process" attribute group. The group name is the first part of a device variable address:

Process/***Bus***/*Device/Variable*

The names written in italics are defined by the process configuration. The name written in bold is defined by this attribute.

Value	Description
"" (empty)	(Default value) auto name. The attribute group is assigned an automatic name: Process/Busn/... n automatically-generated index number
BusName (not empty)	The attribute group receives the specified name: Process/MyName/... Example: <Bus Name="LMN" ...> Process/LMN/...

Speed optional, fixed value: 921600

DataFormat fixed value: 8N1
Determines the data formal on the serial interface.
Syntax: DataBitsParityBitsStopBits

DataBits: 8 = 8 data bits; 7 = 7 data bits
ParityBits: N = no parity bit; E = even parity; O = odd parity
StopBits: 1 = 1 stop bit; 2 = 2 stop bits

Handshake fixed value for RS485 interfaces: HALF

4.11.2 Device configuration

The device configuration is defined by a "Device" XML element within the bus element and described by the following XML attributes:

```
<External>
   <Bus ...>
      <Device _="ID" Name="Alias" MS-ID="MS_ID" PollRate="Rate
       TUnit" >
               <<< Configuration of the device variables >>>
      </Device>
   </Bus>
</External>
```

A "Device" entry must be configured for each meter that is connected to the interface (COM1 or COM2) and that is to communicate with the FP IoT gateway.

Description of the *(italic and bold)* attribute values*:*

ID, required, numbers 0 - 255

Meter bus ID.
It is used for differentiation if several devices are configured.

MS_ID required, measuring point number that is printed on the meters. Numbers, 8-digit meter communication address.

Alias optional, default value: “” (empty), maximum of 20 alpha-numeric characters, must not begin with a number.

Defines the name of the TiXML attribute group that represents the device in the "Bus” attribute group. The group name is the second part of a device variable address:

Process/*Bus*/***Device***/*Variable*

The names written in italics are defined by the process configuration. The name written in bold is defined by this attribute.

Value	Description
“” (empty)	(Default value) auto name. The attribute group is assigned an automatically-generated name: Process/*BusName/*D*Id…* *Id* Value of the “_” attribute
DeviceName (not empty)	The attribute group receives the specified name: Process/*Bus/DeviceName…* Example: <Bus Name=”LMN” ….> <Device Name=”Meter1” ….> Process/LMN/Meter1/…

Rate optional, default: “15m”, for valid values, see below
Poll rate in time units (see TUnit), with which the FP IoT gateway queries the device.
For value ranges, see "BusPollrate"

TUnit Time unit (ms, s, m, h)

4.11.3 Device variable configuration

The configuration for a device variable is defined by an XML element within the "Device" element to which the device variable belongs. The following XML attributes determine the configuration for a variable:

```
<External>
  <Bus …>
    <Device ...>
      <ValueName  _="Bool"   obis="OBIS" format="Format" />
      <ValueName  _="Count"  obis="OBIS" format="Format" exp="Exp"
                             multip="Factor" />
      <ValueName  _="SCount" obis="OBIS" format="Format" exp="Exp"
                             multip="Factor"/>
      <ValueName  _="ASCII"  size="Size" obis="OBIS" format="Format" />
    </Device>
  </Bus>
</External>
```

Each element describes a device parameter value that can be read by the meter in order to generate alarm messages or log the device status for example.

There are four classes of device variables. One class combines all device parameters that have the same properties (storage, encoding, reading or writing possible).

The value for the type attribute "_" (highlighted in red) defines the class to which a device variable belongs:

Variable class	Description
Bool	Parameter value represented by a boolean value The value for a "Bool" device variable is saved in the FP IoT gateway as a bit (simpleType="Bit"). The "obis" attribute indicates the OBIS code. *Example* Reading the "Output a manufacturer-specific data set on the INFO interface" setting `<Fbz_0078 _="Bool" obis="94.49.1*2" />`
Count	Meter value represented by a fixed-point number in OBIS format. The value for a "Count" device variable is saved in the FP IoT gateway as an unsigned double precision floating point number (simpleType="Double"). **Note:** Count variables are intended for saving meter readings. The meter readings contain up to 10 decimal places. This type of number cannot be saved in a DWORD integer value (maximum of 9 decimal places). Double values provide the required precision (maximum of 16 decimal places). As only one integer arithmetic is available for further calculations in the FP IoT gateway, the "precision" attribute must be used to convert the double value into an integer value. Precision losses may occur.

	The "**obis**" attribute indicates the OBIS code.  *Example* **Reading the meter for active energy in the +A direction** `<Fbz_0098 _="Count" obis="1.8.0*255" />`
SCount	Meter value represented by a fixed-point number in OBIS format. The value for a "Count" device variable is saved in the FP IoT gateway as a signed double precision floating point number (simpleType="**Double**"). **Note:** Count variables are intended for saving meter readings. The meter readings contain up to 10 decimal places. This type of number cannot be saved in a DWORD integer value (maximum of 9 decimal places). Double values provide the required precision (maximum of 16 decimal places). As only one integer arithmetic is available for further calculations in the FP IoT gateway, the "**precision**" attribute must be used to convert the double value into an integer value. Precision losses may occur. The "**obis**" attribute indicates the OBIS code.  *Example* **Reading the current active power** `<Fbz_0103 _="SCount" obis="1.7.0*255" />`
ASCII	Parameter value represented by an ASCII string. The "**size**" attribute defines the string in bytes. The "**obis**" attribute indicates the OBIS code.  *Example* **Reading the manufacturer name** `<Fbz_0084 _="ASCII" size="20" obis="96.50.1*1" />`

Each class has its own set of attributes.

Description of the (italic and bold) attribute values:

ValueName Device variable name, maximum of 30 characters, only alphanumeric characters, no umlauts, must not begin with a number. Forms the last part of the device variable address:

Process/*Bus*/*Device*/***VariableName***
The names written in italics are defined by the process configuration.
The name written in bold is defined by this value.

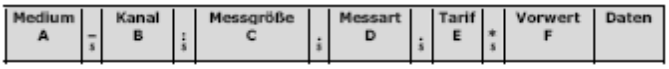

Example

```
<Bus Name="LMN" … >
   <Device Name="Meter1" … >
      <Cnt_Counter…../>
```

Process/LMN/Meter1/Cnt_Counter

OBIS required, OBIS code
OBIS code to address the data point in abbreviated OBIS format.

The code has the following layout:



Fields A+B must be omitted and are set internally to “1-0” automatically. Field F is optional.
If field F is omitted, it is replaced by 255 (=FF) internally.

Size required, 0 - 100
Meaning only for variables in the ASCII class. Describes the maximum number of ASCII characters in OBIS format.

Format optional, the default value is “” (no formatting).
The format options designate the default formatting for the value as it is output, e.g. in e-mails or in the Get command (see the "TiXML Reference Manual").

Factor optional, default value: 1/1, format: see below
The value received from the meter (valueDevice) is multiplied by this factor:

valueTDG = Factor * valueDevice
valueDevice = 1/Factor * valueTDG

The factor is written as a break, e.g.: “1/1000” or “3600/1”, the meter and the denominator must be positive whole numbers and must not be 0.

Exp optional, default value: 0, permitted values: see the table below
Exponent of basis 10 that defines the precision of a fixed-point number.
The value saved in the FP IoT gateway is multiplied by 10 exp(*Exp*) (after applying the factor) to form the variable value.

valueVariable = 10^{Exp} * valueGateway.

The exponent therefore defines the position of the decimal point in the fixed-point number.

The following values are possible:

Exp value	Description
-6	Precision = 0.000001
-5	Precision = 0.00001
-4	Precision = 0.0001
-3	Precision = 0.001
-2	Precision = 0.01
-1	Precision = 0.1
0	Precision = 1 (default)
1	Precision = 10
2	Precision = 100
3	Precision = 1000
4	Precision = 10000
5	Precision = 100000
6	Precision = 1000000

Precision optional, default value:0, valid values: see the following table

The value saved in the FP IoT gateway is multiplied by 10 exp(*Exp*) in order to convert the value to integer representation. Integer representation is used when calculating the process variables, e.g using the commands (GT, LT etc.). It therefore specifies the precision when calculating the process variables and must be defined depending on the application.

Integer representation = whole number(10^{Exp} * valueParameter).
The values correspond to the following table.

Precision Value	Description
-6	Factor = 0.000001
-5	Factor = 0.00001
-4	Factor = 0.0001
-3	Factor = 0.001
-2	Factor = 0.01
-1	Factor = 0.1
0	Factor = 1 (default)
1	Factor = 10
2	Factor = 100
3	Factor = 1000
4	Factor = 10000
5	Factor = 100000
6	Factor = 1000000

Sample configuration

```
[<SetConfig _="PROCCFG" ver="y">
<External>

<Bus Name="Bus1" _="COM2" family="LMN" Product="LMN" protocol="LMNProt"
     Mem="120000" baud="921600" handshake="HALF" type="Master" format="8N1"
     AddProperties="Name,TimeStamp">

  <Device Name="MT176" _="1" MS-ID="68575884" Pollrate="5s">
    <Obis_180 _="Count" obis="1.8.0" acc="R" def="0" format=";%% kWh" />
    <Obis_280 _="Count" obis="2.8.0" acc="R" def="" format=";%% kWh" />
    <Obis_1670 _="SCount" obis="16.7.0" acc="R" def="0" format=";%% kWh" />
    <Obis_3270 _="Count" obis="32.7.0" acc="R" def="0" format=";%% V" />
    <Obis_944912 _="Bool" obis="94.49.1*2" acc="R" def="0"/>
    <Obis_965011 _="ASCII" size="10" obis="96.50.1*1" acc="R" def="0"/>
    <Obis_944901 _="Count" obis="94.49.0*1" acc="R" exp="-3" def="0"/>
  </Device>

  <Device Name="MT631" _="2" MS-ID="68956562" Pollrate="2s">
    <Obis_180 _="Count" obis="1.8.0" acc="R" def="0" format=";%% kWh" />
    <Obis_280 _="Count" obis="2.8.0" acc="R" def="0" format=";%% kWh" />
    <Obis_1670 _="SCount" obis="16.7.0" acc="R" def="-1" format=";%% kWh"/>
    <Obis_3270 _="Count" obis="32.7.0" acc="R" def="0" format=";%% V" />
    <Obis_944912 _="Bool" obis="94.49.1*2" acc="R" def="0"/>
    <Obis_965011 _="ASCII" size="5" obis="96.50.1*1" acc="R" def="0"/>
    <Obis_944901 _="Count" obis="94.49.0*1" acc="R" multip="1/10" def="0"/>
    <Obis_944911 _="Bool" obis="94.49.1*1" acc="RW" def="0"/>
  </Device>

</Bus>

</External>
</SetConfig>]
```

Release notes for the current beta version

The following notes apply to beta version v6.0.0.26 (dated: 15/11/2019):

- The following OBIS codes were tested successfully for the MT176:
  94.49.0*1, 94.49.1.*1, 94.49.1.*2, 94.49.1.*3, 94.49.1.*4, 94.49.1.*9, 94.49.1.*10
  96.50.1*1, 1.8.0*255, 1.8.1*255, 32.7.0*255, 52.7.0*255, 72.7.0*255, 16.7.0*255
- The following OBIS codes were tested successfully for the MT631:
  94.49.0*1, 94.49.1.*1, 94.49.1.*2, 94.49.1.*3, 94.49.1.*9, 94.49.1.*10
  96.50.1*1, 1.8.0*255, 1.8.1*255, 32.7.0*255, 52.7.0*255, 72.7.0*255, 16.7.0*255
- Due to a serious error for both meter types, parameters (OBIS codes) are queried individually per parameter, as querying several parameters within a short time causes the meters to go into a condition whereby communication via the LMN interface is no longer possible. This has serious effects on the query rate (the more parameters queried, the longer the entire query takes)
- The firmware enables new query parameters (OBIS codes) to be implemented via a SetConfig command. This interface will be documented in more detail.

4.12 1-wire

The 1-wire bus allows the connection of up to 30 sensors, family=10 or family=28.
We recommend using sensors that are supplied from externally (sensors with 3 connections: GND, VDD, data).
Parasitically supplied operation is not recommended because the bus no longer functions reliably in this operating mode for bus lengths of over 10m.

Configuration takes place via the “External” database:

```
[<SetConfig _="PROCCFG" ver="v">
<External>

  <Bus Name="Bus0" _="1Wire" protocol="1Wire" type="master"
   StrongPullup="enable">

    <Device Name="Device_0" _="0" family="28" serial="000000fbbb1d">
      <Temperature_0 Name="Temp_0" _="DW" simpleType="Int32"/>
    </Device>
    <Device Name="Device_1" _="1" family="28" serial="000000fc8b96">
      <Temperature_1 Name="Temp_1" _="DW" simpleType="Int32"/>
    </Device>

  </Bus>
</External>
</SetConfig>]
```

Specify `"1Wire"`, `type="master"` as the bus protocol.
The individual sensors are configured via device entries. The family (`family=""`) and the serial number (`serial=""`) must be specified.

Note:
Most 1-Wire sensors output the temperature in 1/1000 °C. As of firmware version 5.2.1.6, the **`StrongPullup`** parameter can be used to control the power supply of parasitically controlled sensors (`enable` =on, `disable` =off).
In parasitic operation **`StrongPullup`** should always be set to `enable`. This can improve the reliability of communication with the sensors.

In powered mode the parameter has no effect.

4.12.1 Scanning the 1-wire bus

The `ScanDevices` TiXML command can be used to determine which sensors have been detected on the bus:

```
[<ScanDevices _="1Wire" protocol="1Wire"/>]
```

The command returns all detected sensors in a list view:

```
<ScanDevices>
  <Device Family="10" Serial="000802bdfa08" Value="11240" ExtPower="1" />
  <Device Family="28" Serial="00000556b8d5" Value="15629" ExtPower="1" />
</ScanDevices>
```

As of firmware version 5.2.1.6, the `ScanDevices` command displays the sensor's measured temperature value (`Value`) and the operating mode (`ExtPower` 0 = parasitic, 1 = fed).

The values returned by `ScanDevices` can then be transferred to the external.

```
 [<SetConfig _="PROCCFG" ver="v">
<External>
  <Bus Name="Bus0" _="1Wire" protocol="1Wire" type="master">
    <Device Name="Device_0" _="0" family="10" serial="000802bdfa08">
      <Temperature_0 Name="Temp_0" _="DW" simpleType="Int32"/>
    </Device>
    <Device Name="Device_1" _="1" family="28" serial="00000556b8d5">
      <Temperature_1 Name="Temp_1" _="DW" simpleType="Int32"/>
    </Device>
  </Bus>
</External>
</SetConfig>]
```

Example (Process branch with a configured 1-wire sensor)

```
<Device_0>
  <DeviceState _="1" />
  <ChangeToggle _="0" />
  <Temp_0 _="23456" />
  <ExternalPower _="1" />
</Device_0>
```

The **`ExternalPower`** parameter (0=parasitic, 1=fed) is displayed as of FW version 5.2.1.6.

4.13 Aurora protocol for ABB inverter

The Aurora protocol is a communication protocol for ABB inverters.
It is a simple serial protocol that uses the RS485 interface.
A subset of the protocol is currently supported in “normal mode”.

List of supported Aurora commands (normal mode)

Function	Description
CMD50	Request the state of the system modules
CMD58	Version Reading
CMD59	Request measurement to the DSP, Typ 3 "Grid Power (W)”
CMD63	Serial Number Reading
CMD78	Cumulated energy readings

Configuration takes place via the “External” database and is specific for each Aurora command.

Bus definition

```
<Bus Name="Bus1" _="COM2" family="ABB" Product="Aurora"
     protocol="Aurora,Normal" Mem="120000" baud="19200" handshake="HALF"
     type="Master" format="8N1">
```

4.13.1 CMD50 (”Request state of the system modules“)

Command 50 enables information regarding the inverter’s system modules to be queried.
Several parameters can be read in this function:

```
Global state, Inverter state, DC/DC Channel 1 state,
DC/DC Channel 2 state, Alarm state
```

The parameters in the variable definition are selected using the index (`ind`).

Variable definition

```
<GlobState _="CMD50" ind="ByteIndex" simpleType="Uint8" acc="R"/>
```

`CMD50` = command 50 to read out statuses for the system modules

```
ByteIndex = 0 | 1 | 2 | 3 | 4
0 = Global state, 1 = Inverter state, 2 = DC/DC Channel 1 state,
3 = DC/DC Channel 2 state, 4 = Alarm state
```

The values returned by the command have a different meaning according to the byte index. For more information regarding this, see the Aurora protocol description.

Excerpt from "UAP_00855_AuroraCommunicationProtocol_5_1_6_PUBLIC.pdf":

Global State	Description
0	Sending parameters
1	Wait Sun/Grid
2	Checking Grid
3	Wait Sun/Grid
4	Checking Grid
5	Wait Sun/Grid
6	Checking Grid
7	Wait Sun/Grid
8	Checking Grid
9	Wait Sun/Grid
10	Checking Grid
…	…
151	Global Setting Not Valid
198	System Configuration Error
200	DSP Programming

DcDc State	Description
0	DcDc Off
1	Ramp Start
2	MPPT
3	Not used
4	Input OC
5	Input UV
6	Input OV
7	Input Low
8	No Parameters
9	Bulk OV
10	Communication Error
11	Ramp Fail
12	Internal Error
…	…
255	DcDc Dsp not programmed

Inverter State	Description
0	Stand By
1	Checking Grid
2	Run
3	Bulk OV
4	Out OC
5	IGBT Sat
6	Bulk UV
7	Degauss Error
8	No Parameters
9	Bulk Low
10	Grid OV
46	Grid Fail
47	Input OC
…	…
255	Inverter Dsp not programmed

Alarm State	Description	Code
0	No alarm	
1	Sun low	W001
2	Input OC	E001
3	Input UV	W002
4	Input OV	E002
5	Sun low	W001
6	No parameters (DSP)	E003
7	Bulk OV	E004
8	Internal error	E005
9	Output OC	E006
10	IGBT Sat	E007
…	…	
157	System frozen	W058
158	Output power OL	W059
159	Wrong wiring	E089

Example *(Read out all inverter statuses using CMD50)*

```
<GlobState      _="CMD50" ind="0" simpleType="Uint8" acc="R"/>
<InvState       _="CMD50" ind="1" simpleType="Uint8" acc="R"/>
<DCDCChn1State  _="CMD50" ind="2" simpleType="Uint8" acc="R"/>
<DCDCChn2State  _="CMD50" ind="3" simpleType="Uint8" acc="R"/>
<AlarmState     _="CMD50" ind="4" simpleType="Uint8" acc="R"/>
```

Process data query

```
<Get _="/Process/Aurora/ABB_1/GlobState" ver="y" />
<Get _="/Process/Aurora/ABB_1/InvState" ver="y" />
<Get _="/Process/Aurora/ABB_1/DCDCChn1State" ver="y" />
<Get _="/Process/Aurora/ABB_1/DCDCChn2State" ver="y" />
<Get _="/Process/Aurora/ABB_1/AlarmState" ver="y" />
```

Result

```
<GlobState _="1" />
<InvState _="0" />
<DCDCChn1State _="0" />
<DCDCChn2State _="7" />
<AlarmState _="0" />
```

4.13.2 CMD58 (“Version Reading“)

Several parameters can be read in this function:
`Model identifier, actual supported grid-standard, transformer, type`

The parameters in the variable definition are selected using the index (`ind`).

Variable definition

```
<Model _="CMD58" ind="ByteIndex" simpleType="String" size="1" acc="R"/>
```

`CMD58` = command 58 to read out version information

```
ByteIndex = 1 | 2 | 3 | 4
 1 = model, 2 = grid, 3 = transformer, 4=type
```

The values returned by the command have a different meaning according to the byte index. For more information regarding this, see the Aurora protocol description.

Excerpt from “UAP_00855_AuroraCommunicationProtocol_5_1_6_PUBLIC.pdf”:

Model (excerpt)

Model char	Model code	Indoor/Outdoor Type
'i'	105	PVI-2000
'o'	111	PVI-2000-OUTD
'I'	73	PVI-3600
'O'	79	PVI-3600-OUTD
'5'	53	PVI-5000-OUTD
'6'	54	PVI-6000-OUTD
'P'	80	3-phase-interface 3G74
'C'	67	PVI-CENTRAL-50 module
'4'	52	PVI-4.2-OUTD
'3'	51	PVI-3.6-OUTD
'2'	50	PVI-3.8-OUTD
…	…	
'9'	57	UNO-3.6-TL-OUTD
	236	UNO-3.8-TL-OUTD
'x'	120	UNO-4.2-TL-OUTD

Grid (excerpt)

Grid char	Grid code	Grid Standard
'A'	65	USA – UL1741
'E'	69	Germany – VDE0126
'S'	83	Spain – DR 1663/2000
'I'	73	Italy – ENEL DK 5950
'U'	85	UK – UK G83
'K'	75	Australia – AS 4777
'F'	70	France – VDE French Model
'R'	82	Ireland – EN50438
'B'	66	Belgium – VDE Belgium model
'O'	79	Korea
'G'	71	Greece – VDE Greece model
…	…	…
'!'	33	Hawaii – 208 V Single Ph.
'”'	34	Hawaii – 240 V Split Ph.
'#'	35	Hawaii – 277 V Single Ph.

Transformer

Transformer	Beschreibung
'N'	Transformerless version
'T'	Transformer version
't'	Transformer HF version
'x'	Dummy transformer type

Type

Transformer	Beschreibung
'N'	Photovoltaic version
'W'	Eolic version
'x'	Dummy inverter type
'n'	Photovoltaic version + ESS (energy storage system)
'w'	Eolic version + ESS (energy storage system)

Example *(Read out all inverter version information using CMD58)*

```
<Model        _="CMD58" ind="1" simpleType="String" size="1" acc="R"/>
<Grid         _="CMD58" ind="2" simpleType="String" size="1" acc="R"/>
<Transformer  _="CMD58" ind="3" simpleType="String" size="1" acc="R"/>
<Type         _="CMD58" ind="4" simpleType="String" size="1" acc="R"/>
```

Process data query

```
<Get _="/Process/Aurora/ABB_1/Model" ver="y" />
<Get _="/Process/Aurora/ABB_1/Grid" ver="y" />
<Get _="/Process/Aurora/ABB_1/Transformer" ver="y" />
<Get _="/Process/Aurora/ABB_1/Type" ver="y" />
```

Result

```
<Model _="4" />       -> PVI-4.2-OUTD
<Grid _="E" />        -> Germany - VDE0126
<Transformer _="T"    -> transformer version
<Type _="N" />        -> photovoltaic version
```

4.13.3 CMD59 ("Request Measurement to the DSP")

Command 59 enables DSP measured values to be read out. A type parameter can be used to read out numerous DSP values.

Variable definition

```
<DSPMeas _="CMD59" type="DSPType" global="1" simpleType="float" acc="R"/>
```

`CMD59` = command 59 to read out measured values

`DSPType = 1 - 203` (for the meaning of the type variables, see the Aurora specification)

The values returned by the command have a different meaning according to the DSPType. For more information regarding this, see the Aurora protocol description.

Excerpt from "UAP_00855_AuroraCommunicationProtocol_5_1_6_PUBLIC.pdf":

DSPType	Measured value	Supported devices	Remarks
1	Grid Voltage (V)	All	If 3-phase = average value
2	Grid Current (A)	All	If 3-phase = average value
3	Grid Power (W)	All	If 3-phase = average value
4	Frequency (Hz)	All	If 3-phase = average value
…	…	…	…
201	Self Consumption Total	REACT	New architecture ABB inverter
202	Self Sufficiency Today	REACT	New architecture ABB inverter
203	Self Sufficiency Total	REACT	New architecture ABB inverter

Attention:
The "PVI-CENTRAL-350 Liquid Cooled AC GATHERING" inverter types are not supported.

Example *(Read out the inverter grid power using CMD59)*

```
<GridPower _="CMD59" type="3" global="1" simpleType="float" acc="R"/>
```

Process data query

```
<Get _="/Process/Aurora/ABB_1/GridPower" ver="y" />
```

Result (value in [W])

```
<GridPower _="159.23" />
```

4.13.4 CMD63 ("Serial Number Reading")

Command 63 reads the inverter's serial number out.

Variable definition

```
<SerialNumber _="CMD63" simpleType="String" size="6" acc="R"/>

CMD63 = command 63 to read out the serial number
```

Process data query

```
<Get _="/Process/Aurora/ABB_1/SerialNumber" ver="y" />
```

Result

```
<SerialNumber _="123456" />
```

4.13.5 CMD78 (“Cumulated Energy Readings“)

Command 78 reads the cumulated energy values in [Wh] for various periods out.

Variable definition

```
<CumEnergyRead _="CMD78" type="Type" global="1" simpleType="Uint32" acc="R"/>
```

`CMD78` = command 78 to read out measured values

`Type = 0 .. 204` (Bedeutung der Type-Variablen siehe Aurora-Spezifikation)

The values returned by the command have a different meaning according to the type. For more information regarding this, see the Aurora protocol description.

Excerpt from “UAP_00855_AuroraCommunicationProtocol_5_1_6_PUBLIC.pdf”:

Slot 0:

Type	Description
0	Daily Energy
1	Weekly Energy
2	Not used
3	Monthly Energy
4	Yearly Energy
5	Total Energy (total lifetime)
6	Partial Energy (cumulated since last reset)

Slot 1:

Type	Description
100	Daily Energy
101	Weekly Energy
102	Not used
103	Monthly Energy
104	Yearly Energy
105	Total Energy (total lifetime)
106	Partial Energy (cumulated since last reset)

Slot 2:

Type	Description
107	Daily Energy
108	Weekly Energy
109	Not used
110	Monthly Energy
111	Yearly Energy
112	Total Energy (total lifetime)
113	Partial Energy (cumulated since last reset)

Slot 3: Type = 114 - 120

Slot 4: Type = 121 – 127

Slot 5: Type = 121 – 127

. . .

Slot 14: Type = 191 - 197

Slot 15: Type = 198 - 204

Example: Variable definition (all variables for slot 0)

```
<DailyEnergy    _="CMD78" type="0" global="1" simpleType="Uint32" acc="R"/>
<WeeklyEnergy   _="CMD78" type="1" global="1" simpleType="Uint32" acc="R"/>
<MonthlyEnergy  _="CMD78" type="3" global="1" simpleType="Uint32" acc="R"/>
<YearlyEnergy   _="CMD78" type="4" global="1" simpleType="Uint32" acc="R"/>
<TotalEnergy    _="CMD78" type="5" global="1" simpleType="Uint32" acc="R"/>

CMD63 = command 63 to read out the serial number
```

Process data query (slot 0)

```
<Get _="/Process/Aurora/ABB_1/DailyEnergy" ver="y" />
<Get _="/Process/Aurora/ABB_1/WeeklyEnergy" ver="y" />
<Get _="/Process/Aurora/ABB_1/MonthlyEnergy" ver="y" />
<Get _="/Process/Aurora/ABB_1/YearlyEnergy" ver="y" />
<Get _="/Process/Aurora/ABB_1/TotalEnergy" ver="y" />
```

Result (all values in [Wh])

```
<DailyEnergy _="1821" />
<WeeklyEnergy _="5290" />
<MonthlyEnergy _="19467" />
<YearlyEnergy _="252178" />
<TotalEnergy _="1325678" />
```

5 Formatting PLC variable values

Without formatting, the variables are displayed as they are transmitted by the PLC. The FP IoT gateway can change these values into number formats and replace boolean variables with strings. The formatted variable is included in e-mails and output in Get commands (without their own format attribute).

Formatting PLC variable values	
Syntax	*When defining the variables:* **<Variable ...simpleType="Uint8" exp="2"... format="*Elements;Text*"/>** *When querying the variable values:* **<Get ... format="*Elements;Text*"/>** *When setting the variable value:* **<Set ... format="*Elements*"/>**
Description:	The `format` parameter comprises two parts that are separated by a **semicolon**: **1st part:** Contains **format elements** that describe outputting or inputting numbers. Apart from the thousand separator "T" and the number format "F", the format elements cannot be combined. The thousand separator element can be positioned anywhere in the format command. The format application depends on the type of variables used. Not all formatting options are equally suitable for all types. The variable's basic type set in the "simpleType" attribute of the variable definition is decisive for a formatting element's suitability. Therefore, the basic types that are suitable for each format element are specified here. The first part can also be empty. The variable value is then output in its native format. **2nd part:** Contains a **text** that is output together with the value for the variables. The variable value can be output within this text in the format defined by the first part. To do this, the variable value's position is indicated by a placeholder (%%). Further associated variable values (such as the physical size and the unit) can also be inserted into the text for certain variables using placeholders. The second part can also be omitted. No semicolon has to be put in front in that case. *Example* Both parts "T'F+9,2 ;Radius %% cm" Only 1st part "R16" Only 2nd part "Text with:%% as the value"

Formatting PLC variable values	
Format elements (part 1):	
? – logical alternative	**?string1,string2** This command replaces both values from boolean variables with strings. If the variable is not 0, string1 is output; otherwise, string2. *Usable for the following simpleType values:* Uint8, Uint16, Uint32, Int8, Int16, Int32 with exp= "0"and bit
Example	**<Variable _="F" simpleType="Uint8" exp="0" … format="?open,closed"/>** <Get _="/Process/COM2/D1/Variable"/> FP IoT gateway responds: <Get _="open"/> bei Wert 1
* - select alternative	***Value1:Text1*Value2:Text2**:Text3** This command is used to replace variable values with predefined strings. If the value is Value1, Text1 is output. If the value is Value2, Text2 is output. In all other cases, Text3 is output. * Separator for the value to be detected ** Separator for all other values The number of values is unlimited. *Usable for the following simpleType values:* Uint8, Uint16, Uint32, Int8, Int16, Int32 und exp= "0"
Example	**<Variable _="R" simpleType="Uint8" exp="0" … format="*0:low*1:medium*2:high**:faulty"/>** <Get _="/Process/COM2/D1/Variable"/> FP IoT gateway responds: <Get _="low"/> for value 0 <Get _="medium"/> for value 1 <Get _="high"/> for value 2 <Get _="faulty"/> for value 7
R/r - basic	**Rn/rn** The command defines the basic n for the value to be output. n = 2 Binary output (e.g. 01101010) n = 8 Octal output (e.g. 21057) n = 10 Decimal output (default, e.g. 1234) n = 16 Hexadecimal output (e.g. AE03) Uppercase and lowercase are used to control how letters are output in the hexadecimal representation: R Uppercase is sent (e.g. AE03) r Lowercase is used (e.g. ae03) *Usable for the following simpleType values:* Uint8, Uint16, Uint32, Int8, Int16, Int32 und exp= "0"

Formatting PLC variable values	
Example	Values in uppercase: **<Variable _="R" simpleType="Uint8" exp="0" ... format="R16"/>** <Get _="/Process/COM2/D1/Variable"/> FP IoT gateway responds (variable value=90): <Get _="5A"/> Values in lowercase: **<Variable _="R" simpleType="Uint8" exp="0"... format="r16"/>** <Get _="/Process/COM2/D1/Variable"/> FP IoT gateway responds (variable value=90): <Get _="5a"/>
T - thousand separator	**T*n*** Defines the separator that appears at each thousandth place in the output. n= , Comma as thousand separator (e.g. 12,345,678) n= . Point as thousand separator (e.g. 12.345.678) n= ` Inverted comma as thousand separator (e.g. 12`345`678) n= empty No thousand separator (default) Note: Can be combined with format element "F" but can also be used on its own or omitted. *Usable for the following simpleType values:* Uint8, Uint16, Uint32, Int8, Int16, Int32, Float, Double
Example	**<Variable _="R" simpleType="Uint32" exp="0"... format="T. "/>** <Get _="/Process/COM2/D1/Variable"/> FP IoT gateway responds (variable value=98765): <Get _="98.765"/>
F - number format	**F Sign, spaces, field width, decimal sign, decimal places** This command defines a number's format. It contains various elements that must be specified in the following order: Sign:defines whether a sign is to be output + The sign is always output (e.g. "+12.3" , "-12.3") - Only a negative sign is output (e.g. "12.3" , "-12.3") Spaces: defines how many spaces are to be filled (only when using the field width) 0 Empty spaces are filled with 0 (e.g. 0066.3) empty Empty spaces are not filled (e.g. 66.3) Field width: defines the maximum field size including signs and separators.

Formatting PLC variable values	
	If this entry is not present, the field is unlimited and no spaces are filled. **Always enter values that are large enough here as otherwise, the number that is outputted will be truncated to the left.** **Decimal sign**: This character is used as a decimal separator (optional) , A comma as the separator . A point as the separator (default) **Decimal places**: Defines the number of decimal places. This can be omitted if no decimal sign was specified. **Note:** Can be combined with format element "T" but can also be used on its own or omitted. *Usable for the following simpleType values:* Uint8, Uint16, Uint32, Int8, Int16, Int32, Float, Double
Example	The variable's value is "12345" in all examples: **Sign:** **<Variable _="F" simpleType="Float"… format="F+"/>** <Get _="/Process/COM2/D1/Variable"/> FP IoT gateway responds (value=123.45): <Get _="+123.45"/> **Field width, spaces:** **<Variable _="R" simpleType="Uint32" exp="-3"...** **format="F09"/>** <Get _="/Process/COM2/D1/Variable"/> FP IoT gateway responds (value=123.456): <Get _="00123.456"/> **Fixed number of digits, decimal sign, decimal places, length:** **<Variable ="R" simpleType="Int32" exp="-3"…** **format="T'F+9,2"/>** <Get _="/Process/COM2/D1/Variable"/> FP IoT gateway responds (value=3123.456): <Get _="+3'123,45"/> **<Variable_="R" simpleType="Int32" exp="0"… format="T'F+9.2"/>** <Get _="/Process/COM2/D1/Variable"/> FP IoT gateway responds (value=-3123456): <Get _="-312'345"/>

Formatting PLC variable values	
	Floating point number, decimal sign, decimal places, length: **<Variable _="F" simpleType="Float"… format="T'F+9.2"/>** <Get _="/Process/COM2/D1/Variable"/> FP IoT gateway responds (value=3123.456): <Get _="+3'123.45"/>
Text (part 2): %%	This placeholder indicates the variable value's position in the output text. This part applies to all data types. It is the only formatting option for "String".
Example	**<Variable _="R" simpleType="Int32" exp="-2" … format="F+;Temp: %%°C"/>** <Get _="/Process/COM2/D1/Variable"/> FP IoT gateway responds (value is 123.45): <Get _="Temp: +123.45°C"/>
%M% – M-BUS Medium (VIF) %U% – M-BUS Unit (VIF)	These placeholders output the data for medium and unit transmitted in the M-BUS value information field. Further M-bus placeholders from the data information fields (DIF) (as of FW 5.1.6.8): %N% – Storage number (DIF) %T% – Tariff (DIF) %S% – Subunit (DIF) %F% – Function field (DIF) In order to be able to use this DIF placeholder, in the LOG definition, the applicable variables must be configured with size="17": <D_XX_V_1 _="meterbus" path="/Process/MBus/D_XX/D_XX_V_1" size="17" format=";%%,%U%,%N%,%T%,%S%"/>
Example	<Var01 **simpleType="Int32" exp="-2"... format=";Medium:%M% Wert=%% %U%"**/> <Get _="/Process/COM3/D1/Var01"/> FP IoT gateway responds (variable value=25.30, heat meter volume): <Get _=" Medium:Heat 0 Volume Flow Wert=25.30 l/h"/>

6 Using the PLC variables in the FP IoT gateway

The PLC variables can be used in the FP IoT gateway in the exact same way as the FP IoT gateway's own variables, e.g. to trigger alarms, log data or for telecontrol.

6.1 Default addressing

After defining the PLC variables in the FP IoT gateway, they can be addressed via the process path:

Example (RS232-2, station 0, Alarm11 variable):

```
<L _="&#xae;/Process/COMx/D0/Alarm11"/>
```

The variables are listed in the extension card path (e.g. COM1, COM2, COM3) and the station number (Device ID) for the PLC (e.g. D0, D1, D2, etc.) and can be addressed using their alias names there.

6.2 Addressing via bus name and station name

By using bus names and station names, the variables can be addressed regardless of the interface and station number. Therefore, the PLC interface or station number can be changed in a central location without having to change the entire project.
As of firmware 1.80, the COM port can be defined according to the label with COM1 to COM3.


```
<External>
   <Bus _="COM2" Name="MyPLC" protocol="Mitsubishi,Alpha2"
type="Master"
    baud="9600">
      <Device _="0" Name="AlphaXL" Pollrate="1s">
         <Input1 _="I" ind="1"/>
      </Device>
   </Bus>
</External>
```

Input1 can now be addressed as follows:

```
<L _="&#xae;/Process/MyPLC/AlphaXL/Alarm11"/>
```

This addressing would then also be valid if the "COM2" interface and or DeviceID "0" is changed in the PLC definition.

6.3 Monitoring PLC communication

System variables are inserted automatically for each device for which parameters were set.

DeviceState:

```
[<Get _="/Process/COM?/D?/DeviceState"/>]
```

This variable indicates the current communication state:
The PLC responds: DeviceState=1
No response: DeviceState=0

ChangeToggle:
`[<Get _="/Process/COM?/D?/ChangeToggle"/>]`

If the device detected value changes in the PLC during a poll cycle, this bit variable changes its status.

Both variables can therefore also be used in the EventStates, e.g. as an alarm or log trigger.

Active:
`[<Get _="/Process/COM?/D?/Active"/>]`

This writeable variable enables communication to the PLC to be interrupted:
`[<Set _="/Process/COM?/D?/Active" value="0"/>]` stops communication.
`[<Set _="/Process/COM?/D?/Active" value="1"/>]` starts communication (default).

Index