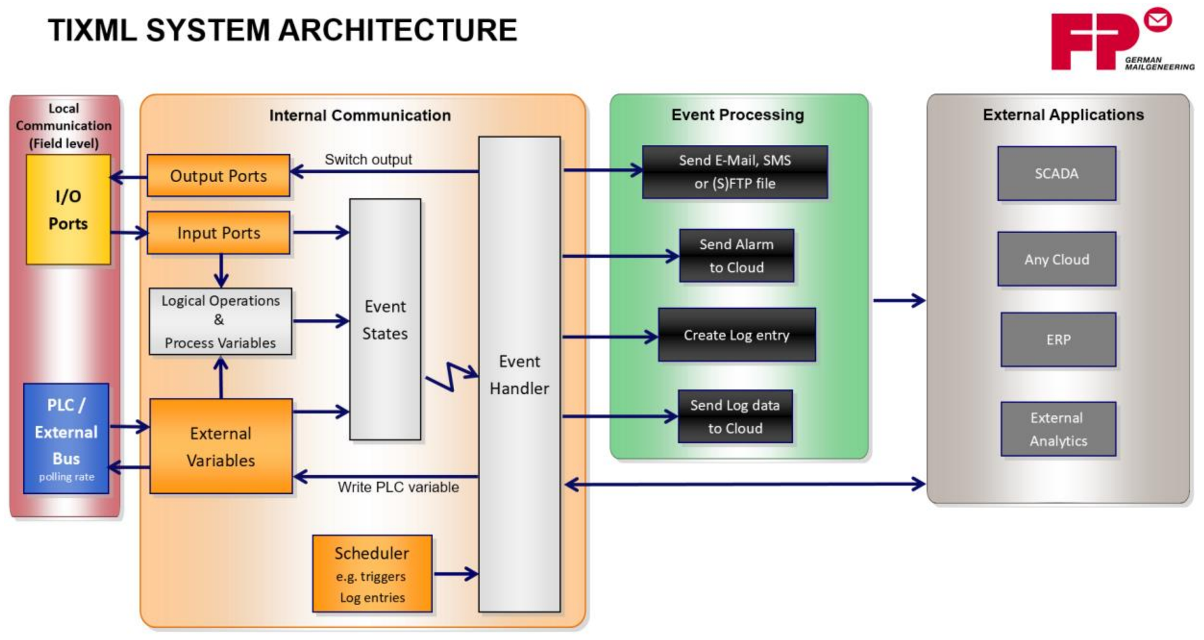
FP INOVOLABS

FP TiXML-Ref-EN

TiXML Reference Manual



For Linux series:	Hx800
	Hx600
	Wx600
	Wx550
For Legacy series:	Hx100/400

Version: 4.4.2

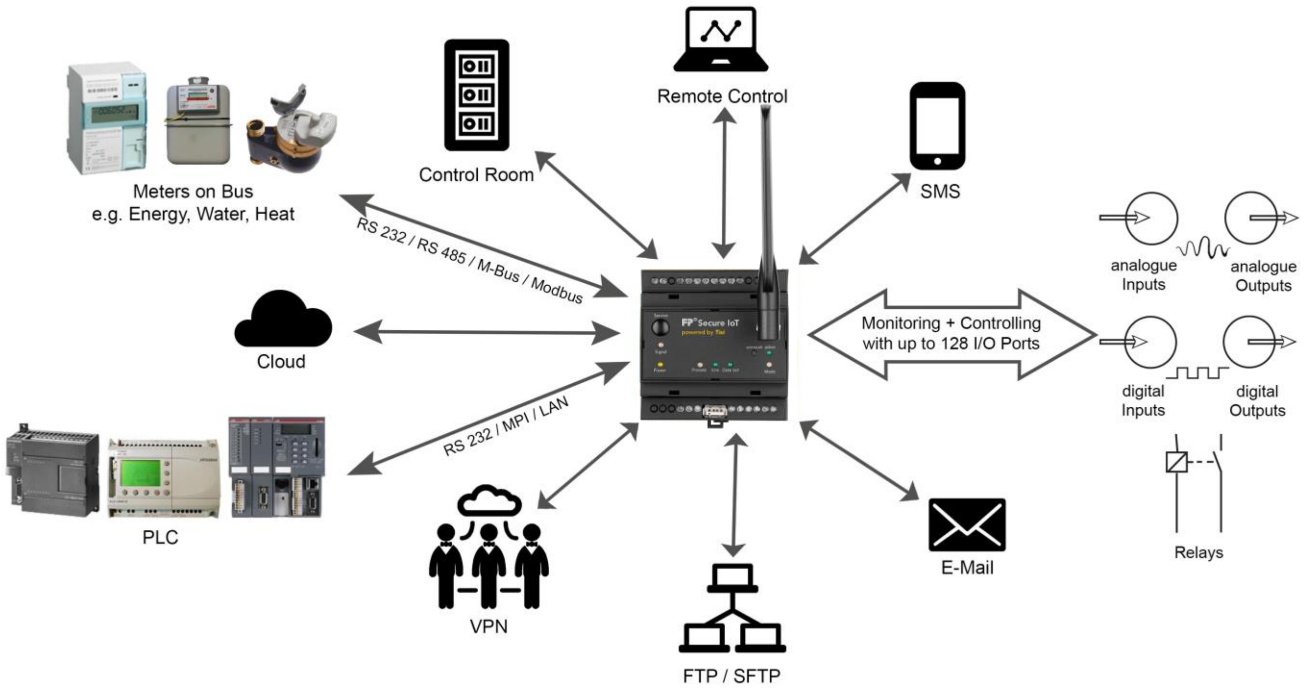
Table of contents

1. Overview

The *FP Gateway* provides a completely new communication type, which can be eaisly integrated into existing systems.

The communication protocols of common PLCs are already implemented into the FP Gateway, so there's no need to change the PLC or its programming. Other PLCs can control the FP Gatewayvia simple text strings: the *TiXML Commands*.

Imagine this as a simple application of FP Gateway:



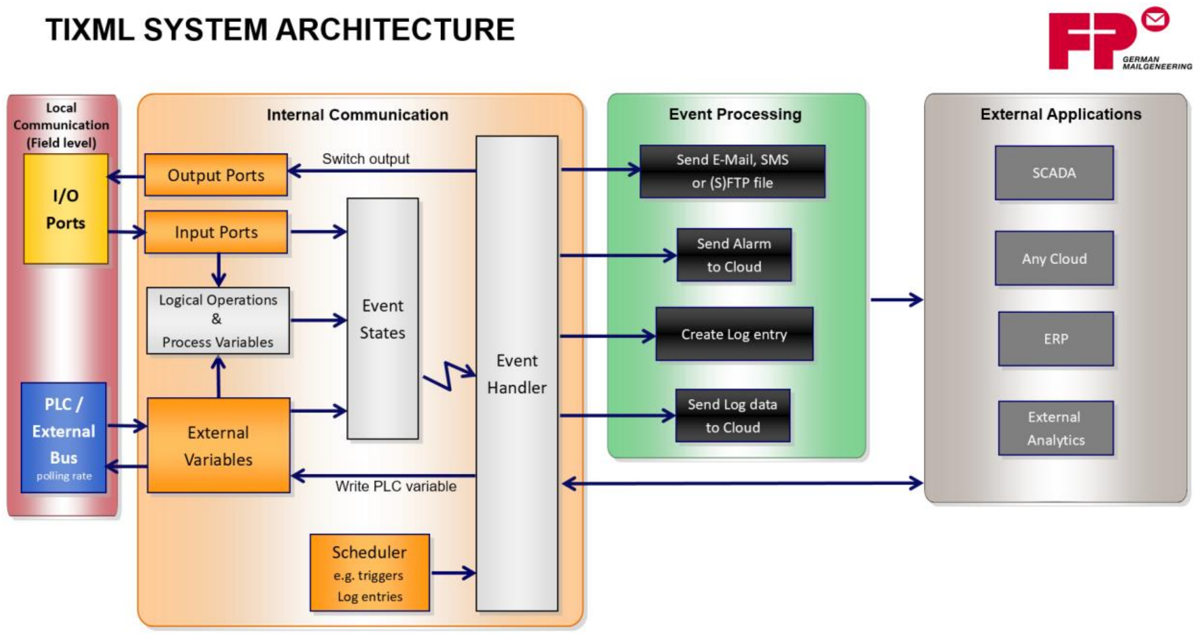
The purpose of this manual is a reference for the features and function s of the FP Gateway.
A step-by-step guide for creating projects can be found in the "TiXML-Tutorial" which is available on our website http://www.inovolabs.com.

The development of the FP Gateway firmware never ends, therefore some of the described features may not be available with your "older" hard- or firmware version. The cover tells you which firmware is required to use all described features (Note "Supported FP Gateway Firmware x.x.x").

See chapter 14.3 for a firmware history.

The TiXML-Reference for devices with firmware version older than 3.0 has the manual code "TiXML-EN" and is also available on our website.

The following picture will help you to understand the processing and relation of all the FP Gateway databases.



At first there is the field level *"Local Communication"* which may be a *PLC* connected via serial port or some switches and measurmenet instruments connected to the FP Gateway *I/O ports*. The configuration of PLCs is documented in a separate "PLC TiXML Manual".

The processing of I/Os or PLC variables is explained in chapter 6. A not supported PLC may control the FP Gateway via TiXML-commands, e.g. the DoOn-Command to activate alarms, see chapter 2.4.6.3

The process *"Event States"* are defining what to do if a variable or I/O-port changes (chapter 6.2). The condition for an event state can be configured in the event state itself, or via *"process variables"* which offer some logical instructions (chapter 6.2).

As soon as an *"Event Handler"* (chapter 3.7) is triggered by an event state, the *"Job Generator"* starts to process the event handler commands e.g. for logging data (chapter 4) or creates a "Sendmail" job using the predefined text templates and address book contacts. A *"Message Job Template"* (chapter 3.9) defines the message type (e.g. SMS) and refers to the recipient from the *address book* (chapter 3.4) and the message templates (chapter 3.8).

Thereafter the *"Job Server"* starts to send the alarm message using the *location* (chapter 3.2) and *user data* (chapter 3.2) settings to calculate the number to dial. For email messages the *Internet Access (ISP)* settings (chapter 3.5) are used to connect to the internet mail servers. For SMS the database of *SMS providers* (chapter 3.10) is used.

2. Controlling an FP Gateway

2.1 Overview

The FP Gateway has a serial interface (RS232) used to control it by a client (control unit, PC, Laptop etc.). The TiXML - Control Protocol is used to control and configure the FP Gateway as a messaging system. TiXML may also be used to control a FP Gateway remotely via a phone line (see chapter 5.2) or via the LAN/Internet (see chapter 3.16).

2.2 RS232 Communication parameters

On the RS232 host port "COM1" TiXML commands must be send with baudrate 115200bps using 1 start bit, 8 databits, none parity, 1 stop bit (8N1).

2.3 Communication Mode

TiXML-Mode uses serial communication at 115200bps with data format 8N1. Hardware handshake (RTS/CTS) is necessary for TiXML communication.
Modem Mode (known from version 2.X) is no longer supported.

TiXML Mode

In this mode the FP Gateway works as a messaging system. A client (for example the control unit or a PC) can now send commands and configurations to the device and can receive responses. You can use a simple terminal program to do this. In this mode the device is responsible for the Simple Tixi Control Protocol (TiXML).

2.4 TiXML - Control Protocol (TiXML)

The Simple Tixi Control Protocol (TiXML) is designed for use of FP Gateway as a messaging system in industrial applications. As clients cannot implement difficult multi-layer protocols like TCP/IP, we decided to create a simple text based protocol that is as easy to use as AT commands. Typically, the client reacts to an event by sending an event message to the FP Gateway. The TiXML protocol is reduced to a minimum so that only the really necessary data about an event is transmitted. Furthermore, the use of a text based protocol makes debugging very simple and the time and effort required for learning and understanding of the protocol is very small.

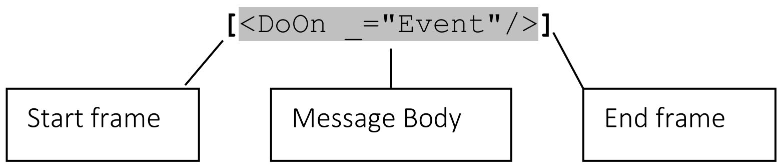
The protocol is derived from Simple Object Access Protocol (SOAP) which is designed for message transports via HTTP and Internet to implement remote procedure calls via the Internet. In TiXML the complete message envelope is replaced by a simple frame "[...]". Only the message contents (body) are used. Like SOAP, the TiXML uses the Extensible Mark-up Language (XML) as the message format. TiXML messages can therefore be edited as XML documents using third party XML editor programs. This reduces the occurrence of syntactical errors.

2.4.1 Overview

TiXML implements a simple text-based remote procedure call mechanism. The client calls a procedure prepared by the FP Gateway (which is in the role of the server) and the FP Gateway answers with a return value (in fact a return message, containing more than one value).To call a procedure, a message is sent by the client to the FP Gateway. The message is enclosed by a message frame (see "Framing"). The procedure and the parameters are encoded as a XML document (see "Command Encoding").

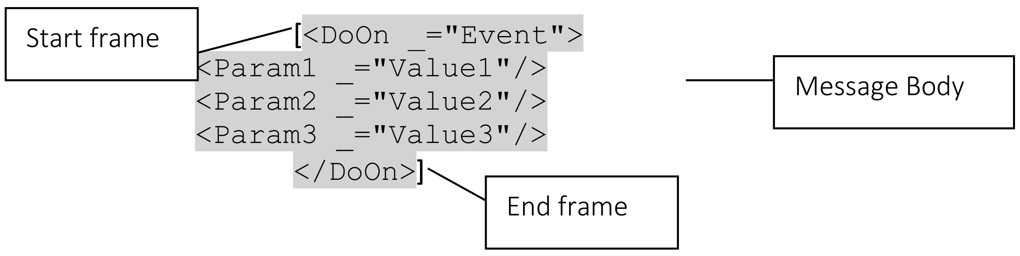
2.4.2 Framing

Each Message is enclosed in square brackets:



```
[<DoOn _="Event"/>]
```

The example shows a message body with one line. Messages with multiple lines are also allowed. Only one <> element is allowed per single line. In this case the message is enclosed in the same way as with one line: the first character is a '[' and the last character in the last line is a closing ']'. No CR/LF is needed at the end of the frame. A line is allowed to consist of maximal 400 characters.



```
          [<DoOn _="Event">
<Param1 _="Value1"/>
<Param2 _="Value2"/>
<Param3 _="Value3"/>
          </DoOn>]
```

The framing is the same for messages send to the FP Gateway as for messages received from the FP Gateway. The FP Gateway does not answer until it has received a complete frame.

Note:
The first two characters of a TiXML command [< have to be entered without delay, otherwise the command will not be recognized.

2.4.3 Command Encoding

Each procedure call and corresponding answer is encoded as a simple XML document. A XML document has a single root element. The name of this element is the name of the called procedure. Both, the procedure call message and the answer message have the same root element name. When the message is sent by the client to the FP Gateway, the message is interpreted as a procedure call. In the opposite direction the message is the answer to a procedure call. Each procedure call is answered by an answering message. If there is an error in the command processing, an error frame is sent by the FP Gateway.

The client should wait with a timeout of 10s for an answer from the FP Gateway before it makes the next procedure call. In some conditions (e.g. creating an email with large Logfile attachment) the response to a DoOn command may even take several minutes.

To generate something like an "AT command set" for controlling a FP Gateway, each procedure is equivalent to a command. Therefore the procedure name is the command name.

The simplest XML document consists of a single element which is also the root element.

`<DoOn _="Event"/>` single element document.

This is equivalent to `<DoOn _="Event"></DoOn>`. As it has no value, the end tag `</DoOn>` can be removed and the tag ends with `/>` instead.

A complex XML document has a tree structure:

```
<DoOn _="Event">
    <Param1 _="Value1"/>
    <Param2 _="Value2"/>
    <Param3 _="Value3"/>
</DoOn>
```

Root Element

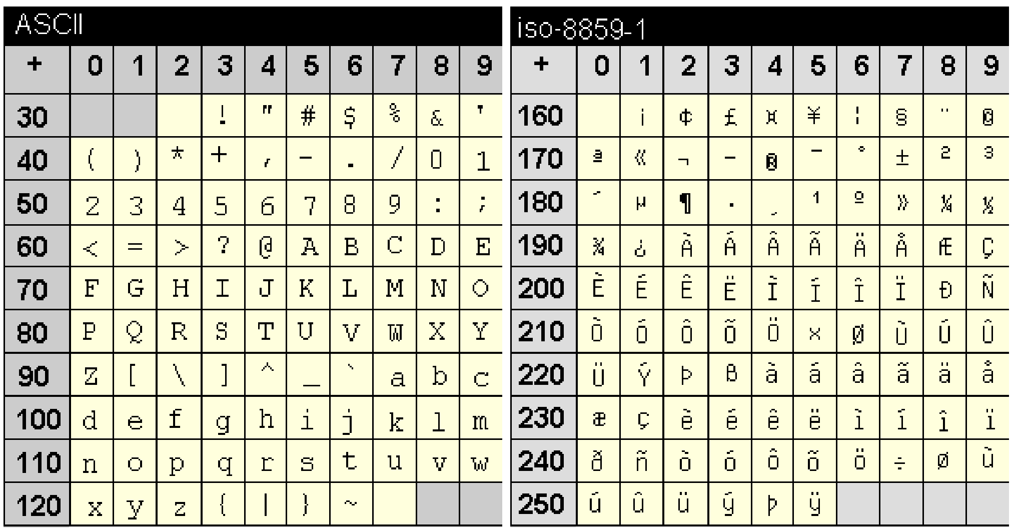
Children of the Root

where the children are enclosed by the tags of the parent.

Only one <> element is allowed per line.

Character Set

TiXML uses character set ISO-8859-1 (ASCII + Latin-1).



ASCII

+	0	1	2	3	4	5	6	7	8	9
30				!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~			

iso-8859-1

+	0	1	2	3	4	5	6	7	8	9
160		¡	¢	£	¤	¥	¦	§	¨	©
170	ª	«	¬	-	®	¯	°	±	²	³
180	´	µ	¶	·	¸	¹	º	»	¼	½
190	¾	¿	À	Á	Â	Ã	Ä	Å	Æ	Ç
200	È	É	Ê	Ë	Ì	Í	Î	Ï	Ð	Ñ
210	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û
220	Ü	Ý	Þ	ß	à	á	â	ã	ä	å
230	æ	ç	è	é	ê	ë	ì	í	î	ï
240	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù
250	ú	û	ü	ý	þ	ÿ				

Some ASCII characters are part of the TiXML syntax, and therefore have to be replaced by "entities" if used inside attribute values:

Character	Entity
<	&lt;
>	&gt;
&	&amp;
“	&quot;

All Latin-1 character (iso-8859-1 supplement) have to be inserted as HEX entity &#x[code];

Character	Entity
ö	&#xf6;
ä	&#xe4;
ü	&#xfc;

The complete code charts can be found here:
http://www.unicode.org/charts/PDF/U0080.pdf

Command Name

The example shows the procedure call message and its answer. Both have the same root element name.

Control Unit sends:
```
[<DoOn _"Alarm_0">
   <Barn _="12"/>
   <Temperature _="10"/>
  </DoOn>]
```

FP Gateway responds:
```
[<DoOn/>]
```

Please note that a command name or tag name must not exceed 20 characters.
Only characters [a-z][A-Z][0-9] and [_] are valid, no digit at the beginning.

Parameters

The parameters of the command and result values can be encoded in different ways:

The command and the answer can have an *owned* parameter. Its name is implicitly known or has the same name as the command. The value of the parameter is encoded as an XML attribute with the character '_' as XML attribute. The attribute value has to be in quotes ("...") (ASCII dec 34) and assigned by an equals sign '='.

Note that there is a *space character between the tag name and the '_' character.*

Additional parameters can be encoded as a *list of Parameters* (pairs of names and values) like in the example above. `<Barn _="12"/>` is a Parameter with the name 'Barn' and the value '12'.

```
[<DoOn _="Alarm_0">
   <Barn _="12"/>
   <Temperature _="10"/>
</DoOn>]
```

Owned Parameter of the Command

Parameter List

In the root element tag some additional parameters can be inserted. These parameters are *optional* and have a default value, which is used when the parameter is not written in the command message.

```
[<DoOn _="Alarm_0" ver="y">
<Barn _="12"/>
<Temperature _="10"/>
</DoOn>]
```

Optional Parameter

Parameter List

Any parameter may include references to system properties (see chapters 3.1.1 and 12):

```
[<DoOn _="Alarm_0">
<Barn _="&#xae;/Process/Bus1/Device_0/Variable_0"/>
<Temperature _="&#xae;/Process/Bus1/Device_0/Variable_1"/>
</DoOn>]
```

References

If you have complex parameters you can encode it as a structured XML document. The following example shows the command to write a complete database where the database is the complex parameter.

```
[<SetConfig _="ISP" ver="y">
<ISP>
   <PPPComm>
      <PPPUserName _="user"/>
      <PPPPassword _="pass"/>
      <AuthentFlags _="3"/>
      <FirstDNSAddr _="194.25.2.129"/>
      <SecondDNSAddr _="193.158.131.19"/>
   </PPPComm>
   <SMTP>
      <mailserver_name _="domain.com"/>
   </SMTP>
   <Modem>
      <RemotePhoneNumber _="+49-30-1234567"/>
      <MediaType _="DATA"/>
      <ModemProtocol _="syncPPP"/>
   </Modem>
</ISP>
</SetConfig>]
```

Complex parameter

Important:
If you send complete projects or large databases to the FP Gateway you need to stop the job processing before sending the first SetConfig using the following command:

```
[<Set _="/Process/Program/Mode" value="Stop" ver="v"/>]
```

After uploading the project/database you have to start the job processing (this is automatically done by FP Gateway reset):

```
[<Set _="/Process/Program/Mode" value="Run" ver="v"/>]
```

2.4.4 Error Frame

When the command processing produces an error, the FP Gateway responds with an error frame. The size of this frame is controlled by the *optional verbose parameter which can be included in each command:*

***ver** - verbose parameter controlling the error response size*	
Syntax	ver="*e*"
Description	This optional parameter can be inserted at each command and controls the size of the error message returned by the FP Gateway.
Elements	*e:* n...short error message — returns an error code *(default)* y...verbose error message — returns a short description v...extended error message — returns an extended verbose description

Example

Short error message:

```
[<GetConfig _="Event/Alert1" ver="n"/>]
  <Error _="-1498"/>
```

Verbose Error Message:

```
[<GetConfig _="Event/Alert1" ver="y"/>]
   <Error>
      <ErrNo _="-1498"/> <ErrText _="path not found"/>
      <ErrorCause> <ErrNo _="-1498"/> <ErrText _="path
       not found"/> <Class _="TXSTCPReadDatabaseCmd"/>
      </ErrorCause>
   </Error>
```

If there are simple microcontroller driven clients, which cannot parse XML data, the default verbose flag 'n' can be used. This returns a single line error frame with an error number. It can be easily processed by this type of device. The verbose flag 'y' should be used during the configuration, when the error frame is not automatically processed but the user needs verbose information on the cause of the error.

For most commands, the following default error frame is created.

Default Error Frame	
Description	Error frame returned by most commands when an error occurs during command processing. This frame is sent instead of the answer frame which is sent when no error occurs. **Note:** Some commands extend this frame by additional classes of errors.

Return	ver="n": `<Error _="errn"/>`
	
	ver="y": `<Error>` *`TiXML Error:`* *`ErrorCause:`* `</Error>`
	
	TiXML Error: Error of the TiXML protocol.
	`<ErrNo _="errn"/>`
	`<ErrText _="Error Description"/>`
	
	ErrorCause: Original error detected in the system.
	`<ErrorCause>`
	`<ErrNo _="errn"/>`
	`<ErrText _="Error Description"/>`
	`<Class _="Class Name"/>`
	`ErrorContext`
	`</ErrorCause>`
	
	ErrorContext: Optional context information on the error.
	`<Context1 _="ContextValue"/>`
	`<Context2 _="ContextValue"/>`
	`<Context3 _="ContextValue"/>`

errn: 0...Error code.

Error Description: Short description text of the error.

Class Name: ID where the error number is related.

ContextValue: The context information.

Example

Short error message:

```
[<GetConfig _="Event/Alert1"/>]
    <Error _="-1498"/>
```

Verbose Error Message:

```
[<GetConfig _="Event/Alert1" ver="y"/>]
    <Error>
      <ErrNo _="-1498"/> <ErrText _="path not found"/>
<ErrorCause> <ErrNo _="-1498"/> <ErrText _="path
        not found"/> <Class _="TXSTCPReadDatabaseCmd"/>
</ErrorCause>
    </Error>
```

2.4.5 Error codes

There are several errors returned or logged if a command or a job could not be processed. The answer frame in this case is a single line error frame. The following error numbers are returned:

Number	*Description*
-24	unable to read from disk
-38	failure to receive expected frame
-43	cannot send EOP frame
-45	disconnection requested by remote
-46	can't transmit end of data
-100	Key not found
-102	last write not yet finished
-102	event contains unknown command
-104	the referenced process variable was invalid
-105	unknown variable type
-105	log file empty on delete
-106	requested path not found
-107	current task already writes to logfile
-107	no event given
-108	unknown error
-109	the syntax of the database is incorrect
-110	the config data is faulty
-112	script terminated with error
-112	no value given
-114	the set command failed
-117	invalid log file content type
-151	required parameter in configuration missing
-200	no address for this station given

-201	template header not found
-202	unknown log file
-204	connection lost
-204	unknown keyword
-204	invalid mail type
-204	the tag name was too long
-205	the requested bus type is not supported
-206	no record given
-207	no memory for mail
-208	temporary no memory for mail
-210	the last write yield an error
-213	no job type given
-215	configuration not found (invalid path)
-219	length of SMS message exceeds 160 chars
-225	no for_all given
-229	copy operation failed
-232	command parameter missing
-239	second StartCheckSum command. Only one is allowed
-240	illegal command parameter (StopCheckSum)
-300	Connection to FP Gateway failed
-306	error writing a read only variable
-307	build up PPP stack failed
-399	Connection to FP Gateway not connected
-401	Frame stream error
-510	file does not exist
-516	select function for send timed out
-520	ordered file space can't be reserved
-607	stack level incomplete on exit
-1095	command is remotely not available
-1097	connection lost - still in remote mode
-1098	authentication required
-1099	command not recognized
-1193	the accessed database is corrupt
-1194	database does not exist
-1195	failed to store database
-1197	could not copy content to file/db
-1199	could not open database
-1496	could not open database
-1497	could not read database
-1498	path not found
-1499	no path given
-1885	the set command failed
-1886	DoOn parameter missing
-1889	could not open journal
-1896	an error in the job creation occurred
-1899	event not in list
-2093	CHAP error - no default user defined
-2094	Connection to FP Gateway lost
-2095	no connection to FP Gateway
-2099	no phone number given
-2194	value exists, but does not contain valid data
-2196	path to key not found

-2197	cannot interpret the value to set
-2297	invalid magic number for factory reset
-2298	reset command remotely not allowed
-2299	invalid reset command
-2391	comport used by configuration tool
-2397	the requested comport does not exist
-2491	no authentication method given
-2493	invalid authentication method
-2497	the user/password is invalid
-2698	error during reading the logfile
-2699	the log entry range is invalid

2.4.6 Commands

TiXML implements several commands (equivalent to the AT command set of a modem) for the control of the device. The commands can be divided into the following groups:

2.4.6.1 Device control

Reset – Reset the device	
Syntax	**`<Reset _="`*`Mode`*`" magic="`*`number`*`"/>`**
Description	This command resets the device. Note: The reset is started when the command is received by the device. Depending on the device (especially Flash Memory Size), 6s to 30s will elapse before the device is ready again.
Parameter	*Mode:* Keep Keep the current settings. Factory Sets the device to its factory settings. Note: All configurations set with the 'SetConfig' command are deleted. GSM and even Ethernet settings (if "persistent") will be kept to guarantee remote access. Update Checks the SD card for a firmware up Download Sets the FP Gateway into firmware update mode. The command has to be followed by ATI9 <CRLF> to detect the upload baudrate (answer: "Serial mode, no modem code!") Thereafter the binary firmware file may be sent using Z-Modem protocol. Format Reformats the flash memory *number:* A magic number is necessary for the factory reset of a remote device or a factory reset via SD card. The magic number is "030406080".
Return	*If no error (command is processed):* `<Reset/>`

	On error (command is not processed): see default error frame (chapter 2.4.4)
Example	Reset the FP Gateway and keep the current settings. *Client sends:* **`[<Reset _="Keep"/>]`** *FP Gateway responds:* **`[<Reset/>]`**

SetTime - Sets the system time of the device	
Syntax	**`<SetTime _="`*`YYYY/MM/DD,hh:mm:ss`*`"/>`**
Description	This command sets the system time of the FP Gateway to the value of the parameter. Note: The FP Gateway contains a Real Time Clock (RTC) that keeps the system time when the power is off. Use this command to set the device time.
Parameter	*YYYY:* 1970...2034 – year *MM:* 01...12- month *DD:* 01...31- day *hh:* 00...23- hour *mm:* 00...59- minutes *ss:* 00...59- seconds
Return	*If no error (command is processed):* `<SetTime/>` *On error (command is not processed):* see default error frame (chapter 2.4.4)
Example	Set the FP Gateway system time to 18 Aug 2004 13:10 hour and 34 seconds. *Client sends:* **`[<SetTime_="2004/08/18,13:10:34"/>]`** *FP Gateway responds:* **`[<SetTime/>]`**

GetTime - Gets the current system time of the device	
Syntax	**`<GetTime/>`**
Description	This command returns the current system time of the FP Gateway. Note: This command is used by our programming tools to check whether the FP Gateway responds to TiXML commands or not.

Parameter	No Parameter
Return	*If no error (command is processed):* `<GetTime _="`*`YYYY/MM/DD,hh:mm:ss`*`"/>` *YYYY:* 1970...2034 – year *MM:* 01...12- month *DD:* 01...31- day *hh:* 00...23- hour *mm:* 00...59- minutes *ss:* 00...59- seconds *On error (command is not processed):* see default error frame (chapter 2.4.4)
Example	Get the current system time of the FP Gateway. *Client sends:* **`[<GetTime/>]`** *FP Gateway responds:* **`[<GetTime="2004/08/18,13:10:34"/>]`**

TransMode - *Set the FP Gateway to a transparent mode to control the connected client device*	
Syntax	**`<TransMode format="`*`SerialFormat`*`" local="`*`localSerialFormat`*`" baud="`*`Baud Rate`*`" com="`*`comport`*`" handshake="`*`Handshake`*`" keep="`*`time`*`" wait="`*`timeout`*`"/>`**
Description	1. It switches the FP Gateway to a transparent mode. Two modes are possible: - Local transparent mode: Data from COM1 will be routed to the selected extension com port (COM2/COM3). - Remote transparent mode: Data from the local dialing modem will be routed to the selected com port or the host port COM1 (if parameter `com` was omitted) of the remote modem. 2. It transforms the baud rate and the serial data format from the local (COM1 or dialing modem) values to the values that the client device, e.g. PLC, uses. Note: It takes about 100ms to forward the first data after establishing the transparent mode. This transparent mode of the remote FP Gateway is left when the dialup connection is interupted. Thereafter, the remote FP Gateway goes back in the TiXML-Mode. To disconnect a remote transparent mode, the local desktop PC must send the escape sequence (+++) and ATH. A transparent mode to the host port MB (COM1) is denied if a local login session is open. If a local TransMode was established (e.g. from COM1 to COM2), the device goes back to TiXML Mode after DTR-low (after "keep" timeout) on COM1 or after a PnP initialization.

	Remote TransMode is not available for Ethernet devices (FP IoT Gateways).
Parameter	*SerialFormat:* String which encodes the serial format that is used between modem and client device. It has the following syntax (*default "8N1"):* *DataBitsParityBitsStopBits* *DataBits* 8...8 data bits are used 7...7 data bits are used *ParityBits* N...No parity bit E...Even parity O...Odd parity *StopBits* 1...one stop bit 2...two stop bits *localSerialFormat:* Same as "SerialFormat" but used between PC and modem *Baud Rate:* Baudrate in bits per second (bps) (*Default 115200*) *comport:* Specifies the COM port on the FP Gateway used for the connection `COM1` `PLC` port (labeled *COM1 RS232) (default)* `COM2` PLC port (labeled *COM2 RS232 or COM2 R-485/422)* (if available) `COM3` M-Bus interface (labeled *M-Bus*) (if available) *Handshake:* Used communication handshake. `None` communication without handshake `XONXOFF` software handshake `XONXOFFPASS` software handshake, XONXOFF forwarded to application `RTSCTS` hardware handshake with RTS CTS `DTRDSR` hardware handshake with DTR DSR `HALF` Halfduplex RS 485 communication `FULL` Fullduplex RS 485/422 `HALFX` Halfduplex RS 485 communication with XON XOFF `FULLX` Fullduplex RS 485/422 with XON XOFF `noDTR` disables DTR control, DTR will be always active

*time:*There are two different functions for this parameter, depending on the connection:

During connection from one FP Gateway port to its second port:
Specifies the time period the FP Gateway will wait for the application to take over the serial port *(Default: 0s*). After this time, the FP Gateway will automatically leave the TransMode.

During remote connection (optional):
Specifies the time period the FP Gateway will try to disable the bus protocol at the specified com port to establish a transparent connection.

timeout: Specifies the time the FP Gateway will try to disable a PLC bus protocol on the remote com port *(Default: 20s).*

Return

This command returns no TiXML frame because the TiXML protocol is left.
The string **`CONNECT`** acknowledges the established transparent mode.

Example

1. Set the FP Gateway to the transparent mode and connect it to the client device with 57600bps and 8 data bits, even parity bit and one stop bit. Data from COM1 will be routed to COM2 vice versa:

Client sends:
```
[<TransMode baud="57600" local="8E1" format="8E1"
com="COM2"/>]
```

FP Gateway responds:
```
CONNECT
```

2. Connect to a remote FP Gateway, switch it to the transparent mode with a baud rate of 9600 and a format of 8N1 on RS 485 halfduplex interface COM2.

Client sends:
```
[<TransMode baud="9600" format="8N1" com="COM2"
handshake="HALF" keep="20s"/>]
```

FP Gateway responds:
```
CONNECT
```

Steps after CONNECT message:
1. Disconnect the client (R-CON, TILA2, TICO) from COM Port.
2. Connect the other control program (e.g. PLC software) to the COM Port.
3. Control the remote device through the FP Gateway.
4. Disconnect the control program from COM port.
5. Connect the client (R-CON, TILA2, TICO) to the FP Gateway COM port.
6. Disconnect:

Local TransMode:
Client sends: Plug&Play sequence

	Remote TransMode: *Client sends: (wait one second) +++ (wait one second)* *FP Gateway responds:* `OK` *Client sends:* `ATH`

ScanWLAN – Scan for available WLAN access points	
Syntax	See chapter 3.15

2.4.6.2 Authentication

Login – Start a TiXML session	
Syntax	**`<Login _="`*`type`*`" user="`*`User Name`*`" password="`*`Password`*`"/>`**
Description	This command starts a TiXML session for a user. By this command, the user authenticates to the device. Note: The FP Gateway can be protected against unauthorized access. In the factory configuration, no access protection is provided and the *Login* command doesn't need to be sent to control the device. In this state each command has its own session, i.e. the session starts implicitly with the command and ends after command return. When the Login command is sent in this state, user name and password are ignored. If you need to upload several databases to the FP Gateway (configuration session), and one of the databases includes user access configuration, it is recommended to send a `[<Login/>]` command at first to open a TiXML session. Otherwise all SetConfig commands after uploading the new user access configuration will be blocked according to the new access protection. To limit the access to certain users, you can create a service/group/user/password map in the configuration of the FP Gateway. If this map is not empty, no command is processed until the login command with a valid user-password pair is sent, for example, by the client. This protection is related to the connection (RS232, LAN or phone line connection) by which the commands are sent. If there is another connection at the same time, this connection needs its own authentication. An accepted login is valid until: • The **`Logout`** command is sent • or a **`Login`** command with an invalid user-password pair is sent • or the power goes off • or the DTR is low • or the remote connection is broken (for remote control only). • or no commands are received for 5 minutes (idle timeout) To create the access protection, see chapter 3.6. There are two ways to remove the access limitation: 1. Send an empty AccRights configureation (see chapter 3.6).

2. Make a factory reset using the **`Reset`** command with the parameter `Factory`. See chapter 2.4.6.1 for details on resetting the FP Gateway. Keep in mind that a factory reset deletes all other settings as well

Parameter

type: PAP (Password Authentication Protocol)
sends the password without encoding.
CHAP (Challenge Handshake Authentication Protocol)
multistep protocol does not send the password, a challenge is exchanged.

User Name User name can be empty if no authentication is required or it must be empty if the default users are configured in the access rights database (password protection).

Password Password. Can be empty.

Return

If no error (command is processed): `<Login/>`

On error (command is not processed): see default error frame (chapter 2.4.4)

Example

Login successful:

Client sends:
```
[<Login _="PAP" user="Name" password="secret"/>]
```

FP Gateway returns:
```
[<Login/>]
```

Login not successful:
Client sends:
```
[<Login _="PAP" user="Name" password="try"/>]
```

FP Gateway returns:
```
[<Error _="-1098"/>]
```

CHAP Login Sequence (Username "Daniel", Plain-Password "test")

Client sends:
```
[<Login _="CHAP" user="Daniel"/>]
```

FP Gateway returns:
```
[<Login _="Challenge">
    <key _="b0a12c96ef9b01f8c07fd98b332c165ffdab5764872ef049" />
   <id _="31" />
</Login>]
```

Client sends:
```
[<Login _="Response" id="31" md5="468041b48c6bca5ffd9834209d8b1935" ver="y"/>]
```

FP Gateway returns:
```
[<Login/>]
```

The hash for the MD5 response is calculated over the string "ID+password+key". The string in the above example would be `"31testb0a12c96ef9b01f8c07fd98b332c165ffdab5764872ef049"`

Note:

	ⓘ A tool for calculating the MD5 hash is not included in the scope of deliverey; you may use one of the many free online calculators, such as http://md5-hash-online.waraxe.us/

Logout – Quit a controlling session	
Syntax	**`<Logout/>`**
Description	This command quits a TiXML session - started with a successful Login. It closes the access to the FP Gateway ⓘ Note: This command can be sent at any time. It affects the access to the FP Gateway if a Login command was sent before. In this case the access right is denied and a new Login will be necessary to get access again. If no access protection is configured or no login session is established, this command does nothing.
Parameter	*No Parameter*
Return	*If no error (command is processed):* `<Logout/>` *On error (command is not processed):* see default error frame (chapter 2.4.4)
Example	Login successful *Client sends:* **`[<Logout/>]`** *FP Gateway returns:* **`[<Logout/>]`**

2.4.6.3 Event processing

DoOn – Trigger an Event	

Syntax	**`<DoOn _="`*`EventName`*`">`**
	`ParameterList`
	`</DoOn>`
	or
	`<DoOn _="`*`EventName`*`"/>`

Description	This starts the processing of a client event. The command can be used to test the processing of events without need to change the associated trigger variables. ⓘ Note: There are two forms of syntax; one is the long form which allows the transmission of additional attributes representing the parameters of the event context. If no event context is present, the short form is used. The command starts the event processing when the result is sent back to the client. In most cases this leads to the creation of message jobs. The sending process of the messages is done while the client can create new event messages. The results of the processing are observed by the FP Gateway itself. The results can be observed by the client using the `ReadLog` command which is described in chapters 2.4.6.6 (how to read logfiles) and

	4 (how to create logfiles). Active jobs can be read using the GetJob command (see following pages)
Parameter	*EventName:* Name of the event to be processed. There must be an EventHandler configuration in the 'EVENTS' database using this name. See chapter 3.7 for details on EventHandler database. *ParameterList:* List of XML encoded parameters representing the event context. A parameter is written in a single XML tag: **`<ParameterKey _="Value"/>`** where *ParameterKey:* is the unique name of the parameter. *Value:* is the value of the parameter.

Return	*If no error (command is processed):* `<DoOn/>`
	
	On error (command is not processed):
	ver="n": `<Error _="errn"/>`
	
	ver="y": `<Error>`
	    `TiXML Error:`
	    `JobGeneratorError`
	    `ErrorCause:`
	  `</Error>`
	
	TiXML Error: Error of the TiXML protocol.
	  `<ErrNo _="errn"/>`
	  `<ErrText _="Error Description"/>`
	
	JobGeneratorError: Error during Job generation.
	  `<JobGeneratorError>`
	    `<ErrNo _="errn"/>`
	    `<ErrText _="Error Description"/>`
	    `ErrorContext`
	  `</JobGeneratorError>`
	
	ErrorCause: Original error detected in the system.
	  `<ErrorCause>`
	    `<ErrNo _="errn"/>`
	    `<ErrText _="Error Description"/>`
	    `<Class _="Class Name"/>`
	    `ErrorContext`
	  `</ErrorCause>`
	
	ErrorContext: Optional context information on the error.
	  `<Context1 _="ContextValue"/>`
	  `<Context2 _="ContextValue"/>`
	  `<Context3 _="ContextValue"/>`
	
	errn:
	    0....OK
	<0...Error code

	Error Description: Short description text of the error. *Class Name:* ID where the error number is related. *ContextValue:* The context information text.

Example	Process the 'TemperatureAlert' event. The event context contains the barn ID = 12 where the temperature is in a critical range and the value of the temperature in degrees of Celsius.
	
	Control Unit sends:
	`[<DoOn _"TemperatureAlert">`
	`<Barn _="12"/>`
	`<Temperature _="10"/>`
	`</DoOn>]`
	
	FP Gateway responds:
	`[<DoOn/>]`

***GetJob** – Shows a list of currently active jobs*	
Syntax	**`<GetJob del="`*`Mode`*`"/>`**
Description	This command shows a list of job groups and currently active jobs. Note: This command may also be used to cancel currently active jobs, e.g. to delete messages from the message queue.
Parameter	*Mode:* y delete all active jobs (jobs can't be deleted if they're currently processed) Mode: n don't delete active jobs (default)

Return	***If no error (command is processed):***
	`<GetJob>` `<`*`JobGroup`*` _="`*`State`*`">` `<Job_`*`X`*`>` `<Time _="`*`Date,Time`*`"/>` `<Type _="`*`Type`*`"/>` `<Priority _="`*`Priority`*`"/>` `<Origin _="Origin"/>` `</Job_`*`X`*`>` `</`*`JobGroup`*`>` …
	`</GetJob>`
	
	JobGroup: Name of job group
	`Modem_Mode`
	`Default` `TSAdapter`
	`SMS_Receive` `SMS_Send` `POP3_Client`
	`HTTP_Server_In`
	`CGI_DoOn`
	`HTTP_Server_Out` `Time_Client` `URL_Send`
	`SMTP_Client`
	`Fax_Send`
	`Text_Fax` `Fax_Receive` `Script_Send`
	`Incoming_Call` `Auto_Transmode`
	`Job_Result_Processor` `Remote_ModemMode`
	`TSAdapterCallback`

State: State of job group

`Started`	Jobs will be processed
`Stopped`	Jobs are not processed (service not licensed)

X: Active job number (increases)

Date,Time: Start time of job

Type: Job type number, e.g. 5=GSMSMS

Priority: Priority of job (see chapter 3.7.1)

Origin: EventHandler name

On error (command is not processed): see default error frame (chapter 2.4.4)

Example

GetJob on idle system state:

Client sends:

```
[<GetJob/>]
```

FP Gateway responds:

```
[<GetJob>
   <Modem_Mode _="Started"/>
   <Default _="Started"/>
   <TSAdapter _="Started"/>
   <SMS_Receive _="Started"/>
   <SMS_Send _="Started"/>
   <POP3_Client _="Started"/>
   <HTTP_Server_In _="Started"/>
   <CGI_DoOn _="Started"/>
   <HTTP_Server_Out _="Started"/>
   <Time_Client _="Started"/>
   <URL_Send _="Started"/>
   <SMTP_Client _="Started"/>
   <Fax_Send _="Started"/>
   <Text_Fax _="Started"/>
   <Fax_Receive _="Started"/>
   <Script_Send _="Started"/>
   <Print_Jobs _="Started"/>
   <Incoming_Call _="Started"/>
   <Auto_Transmode _="Started"/>
   <Job_Result_Processor _="Started"/>
   <Remote_ModemMode _="Started"/>
   <TSAdapterCallback _="Started"/>
</GetJob>]
```

	GetJob with SMS in message queue, waiting for acknowledge:
	
	Client sends:
	`[<GetJob/>]`
	
	FP Gateway responds:
	`[<GetJob>   <Modem_Mode _="Started"/>   …`
	`   <Text_Fax _="Started"/>`
	`   <Script_Send _="Started">`
	`    <Job_3>`
	`      <Time _="2008/04/26,17:25:47"/>`
	`      <Type _="5"/>`
	`      <Priority _="1"/>`
	`      <Origin _="currently unavailable (running)!"/>`
	`    </Job_3>`
	`   </Script_Send>`
	`   <Incoming_Call_Trigger _="Started"/>`
	`   <Job_Result_Processor _="Started">`
	`    <Job_3>`
	`      <Time _="2038/01/19,03:14:07"/>`
	`      <Type _="65"/>`
	`      <Priority _="99"/>`
	`      <Origin _="Alarm_0"/>`
	`    </Job_3>`
	`   </Job_Result_Processor>`
	`   <Remote_ModemMode _="Started"/>`
	`</GetJob>]`
	
	GetJob delete:
	
	Client sends:
	`[<GetJob del="y"/>]`
	
	The FP Gateway answers with the list of all active jobs including the deleted jobs. A thereafter immediately sent [<GetJob/>] will show a list without active jobs (if no new jobs are started in the meantime).

2.4.6.4 Configuration

SetConfig - *Set configuration data*	

Syntax	`<SetConfig _="`***`Path`***`">`
	`   `***`XML-Data`***
	`</SetConfig>`

Description	This command writes configuration data into a database. Note: The FP Gateway stores its data permanently in an embedded XML database. The database is prepared by the firmware of the FP Gateway. It can't be deleted or created by the client. Only the contents of the database can be changed.
Parameter	*Path:* *DataBase/GroupPath* *DataBase* Name of the database where the data has to be written.

	GroupPath Path name in the database where the attribute group is to be inserted/replaced (*optional* and for attribute groups only). ***XML-Data:*** Attribute group or complete XML database document. Note: *Complete attribute groups or databases* can be changed only! The command handler replaces the old attribute group by the new one. Separate attributes of an attribute group cannot be changed. If the attribute group does not exist in the database, the database is extended by the attribute group.
Return	*If no error (command is processed):* `[<SetConfig/>]` *On error (command is not processed):* see default error frame (chapter 2.4.4)

Example	Replace the contents of the 'Modem' attribute group inside the database 'ISP' and the attribute group 'ISP'
	
	Client sends:
	`[<SetConfig _="ISP/ISP">`
	`   <Modem>`
	`      <RemotePhoneNumber _="+49-30-40608117"/>`
	`      <MediaType _="DATA"/>`
	`      <ModemProtocol _="syncPPP"/>`
	`   </Modem>`
	`</SetConfig>]`
	
	FP Gateway responds:
	`[<SetConfig/>]`

Example	Replace the contents of the ISP group inside the database with the name 'ISP'.
	
	Client sends:
	`[<SetConfig _="ISP">`
	`   <ISP>`
	`      <PPPComm>`
	`         <PPPUserName _="user"/>`
	`         <PPPPassword _="pass"/>`
	`         <AuthentFlags _="3"/>`
	`         <FirstDNSAddr _="194.25.2.129"/>`
	`         <SecondDNSAddr _="193.158.131.19"/>`
	`      </PPPComm>`
	
	`      <SMTP>`
	`         <mailserver_name _="domain.com"/>`
	`      </SMTP>`
	`      <Modem>`
	`         <RemotePhoneNumber _="+49-30-1234567"/>`
	`         <MediaType _="DATA"/>`
	`         <ModemProtocol _="syncPPP"/>`
	`      </Modem>`
	`   </ISP>`
	`</SetConfig>]`
	
	FP Gateway responds:
	`[<SetConfig/>]`

GetConfig - Get configuration data	
Syntax	**`<GetConfig _="`*`Path`*`"/>`**
Description	This command reads the configuration data from a database of the FP Gateway.
Parameter	*Path:* *DataBase/GroupPath* *DataBase* Name of the database to read. *GroupPath* Path name in the database to read (*optional* and for attribute groups only).

Return	*If no error (command is processed):*
	`<GetConfig>`
	`   `*`XML-Data`*
	`</GetConfig>`
	
	XML-Data: Sub tree or complete XML database document.
	
	On error (command is not processed): see default error frame (chapter 2.4.4)

Example	Get the contents of the ISP group inside the database with the name 'ISP'.
	
	Client sends:
	`[<GetConfig _="ISP/ISP"/>]`
	
	FP Gateway responds:
	`[<GetConfig>`
	`    <ISP>`
	`      <PPPComm>`
	`         <PPPUserName _="user"/>`
	`         <PPPPassword _="pass"/>`

	`<AuthentFlags _="3"/>`
	`<FirstDNSAddr _="194.25.2.129"/>`
	`<SecondDNSAddr _="193.158.131.19"/>`
	`</PPPComm>`
	
	`<SMTP>`
	`<mailserver_name _="domain.com"/>`
	`</SMTP>`
	`<Modem>`
	`<RemotePhoneNumber _="+49-30-1234567"/>`
	`<MediaType _="DATA"/>`
	`<ModemProtocol _="syncPPP"/>`
	`</Modem>`
	`</ISP>`
	`</GetConfig>]`

2.4.6.5 Process values

***Get** - Get System Property*	
Syntax	**`<Get _="`*`Path`*`" AddInfo="`*`Error`*`" exp="`*`Exponent`*`" multip="`*`Factor+Offset`*`" format="`*`FormatString`*`" ViewProperties="`*`Flags`*`"/>`**
Description	Get the value of the system properties referred to by the *Path* value. The System Properties are the set of data describing a FP Gateway. This includes administrative information like version numbers, licenses etc. which are defined at the creation time of the firmware as well as information on the hardware configuration and the system state. The system state includes the system time, the system mode, the states of the I/O ports and PLC variables etc. The configuration settings defined by the SetConfig are a part of the system state and therefore a part of the system properties. They can therefore also be accessed by the Get command. The difference to the GetConfig command is the way the data is addressed and the structure of the returned data. Both commands use a slash separated path to address the data but Get addresses a single value only where GetConfig addresses complete attribute groups. A second difference is in the data itself. All System Properties have a unique address defined by their path. Configurations contain parts which have no unique addresses: For example the PLC "External" could contain several "Devices" on a "Bus" which all have the same tag name "Device". In this case an element can't be addressed uniquely by a path. Therefore, not all elements of the configuration can be addressed by the Get command. Use GetConfig instead.
Parameters	*empty*: If no parameter is given, the FP Gateway will send a list of all system properties. *Path:* Path which addresses the system property. See chapter 14 for details on system properties. The last element of the latter may be a variable name or the name of a system property tree followed by a slash. *Error :* For directly reading the error state of a process variable or PLC variable, the "Get" command may be extended by the AddInfo attribute that displays the value of an additional information ("`ErrorClass,ErrorValue`") instead of displaying the variable value ErrorClass:

0 = no error
1 = error (see ErrorValue)
ErrorValue:
See chapter 6.3.2 for ErrorValues.

See PLC-TiXML-Manual for further information.

Exponent:
Exponent of base 10 to specify fix point precision of
simpleType = Uint8, Uint16, Uint32, Int8, Int16, Int32 (see chapter 6.5.1*)*.
The process variable value will be multiplied by 10 exp(*Exp*) to get the parameter value.

valueParameter = 10^{Exp} * value process variable.

The exponent therefore specifies the position of comma within a fix point value
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

	Factor+Offset The value will be multiplied by this factor and the offset will be added to get the output value. simpleType = Uint8, Uint16, Uint32, Int8, Int16, Int32 (see chapter 6.5.1*).* Output value = Factor * Logged value + Offset The factor is used as a fraction, e.g.: "1/1000" or "3600/1", the denominator and numerator must not be zero. The offset may be negative or positive. *FormatString:* **integer:** (for PLC and process variables) 1. *simpleType = Uint8, Uint16, Uint32, Int8, Int16, Int32* (see chapter 6.5.1) The value is displayed as integer. The integer calculates itself by using the exponent specified with the variable and its value : *Value as integer = 10* $^{-Exp}$ ** value* 2. *simpleType = float, double* (see chapter 6.5.1) The value is displayed as integer. The integer calculates itself by using the precision specified with the variable and its value: *Value as integer = 10* $^{-Precisoin}$ ** value* *3. all other data types* The value is displayed native (see chapter 6.5.1) **native: (or empty)** The value is displayed native. (see chapter 6.5.1) **FormatString:** String that defines the value output format. For a list of available format option see chapter 6.5. *Flags:* Additional informations of external device variables (tree /Process/BusX/) will be displayed: **Name:** Variable alias name if attribute `Name` is specified within External database. See PLC TiXML manual for more details. **TimeStamp:** Time stamp of the last successful polling of an external device variable ***Error:*** The error state of a process variable or PLC variable, For the structure of the error state, see the "AddInfo" attribute of this command.
Return	*If no error (command is processed):* `<Get _="`*`value`*`"/>` *value:* value of the system properties *On error (command is not processed):* see default error frame (see chapter 2.4.4)

Example

Get the complete FP Gateway system properties:
Client sends: **[<Get ver="y"/>]**

Get the state of all digital inputs of the mainboard:
Client sends: **[<Get _="/Process/MB/IO/I/"/>]**
FP Gateway responds:
```
[<Get>
  <I>
<P0 _="1"/>                                 <P1 _="1"/>
  </I>
</Get>]
```

Get the state of the first digital input of the mainboard:
Client sends: **[<Get _="/Process/MB/IO/I/P0"/>]**
FP Gateway responds: **[<Get _="1"/>]**

The status of the FP Gateway digital I/Os can also be read as a byte, word or dword value by adding B (byte), W (word) or D (dword) to the branch "I" within the system property path and referring to the first port within the group (see chapter 13.1):
Get mainboard inputs as a byte value:
Client sends: **[<Get _="/Process/MB/IO/IB/P0"/>]**
FP Gateway responds: **[<Get _="3"/>]**

Get mainboard inputs as a dword value:
Client sends: **[<Get _="/Process/MB/IO/ID/P0"/>]**
FP Gateway responds: **[<Get _="3"/>]**

Get the FP Gateway serial number:
Client sends: **[<Get _="/SerialNo" ver="y"/>]**
FP Gateway responds: **[<Get _="00081101"/>]**

Get and reformat the state of the analog input of the mainboard with two decimal places:
Client sends: **[<Get _="/Process/MB/A/AI/P0" format="F,2"/>]**
FP Gateway responds: **[<Get _="31,45"/>]**

Get the state of the analog input of the mainboard with an offset of 50:
Client sends: **[<Get _="/Process/MB/A/AI/P0" multip="1/1+50"/>]**
FP Gateway responds: **[<Get _="3195"/>]**

Get the native value of a formatted process variable:
Client sends: **[<Get _="/Process/PV/Variable" format="native"/>]**
FP Gateway responds: **[<Get _="12345"/>]**

Get the error state of the PLC variable "Variable_0" at "Device_0" on PLC-bus "Bus1":
Client sends: **[<Get_="/Process/Bus1/Device_0/Variable_0" AddInfo="Error"/>]**
FP Gateway responds: **[<Get _="0,0"/>]**

	Get the alias name, the error state and the last successful polling of the PLC variable "Variable_0" at "Device_0" on PLC-bus "Bus1": *Client sends:* **`[<Get _="/Process/Bus1/Device_0/Variable_0" ViewProperties="Name,Error,TimeStamp"/>]`** *FP Gateway responds:* **`[<Get> <Variable_0 _="3.373" Name="Energie" Error="0,0" TimeStamp="2009/01/22,14:03:09"/> </Get>]`** Get the maximal dial attempts defined in the USER database USER section: *Client sends:* **`[<Get _="/USER/USER/MaxDialAttempts"/>]`** *FP Gateway responds:* **`[<Get _="2"/>]`**

Set *- Set System Properties*	
Syntax	**`<Set _="`*`Path`*`" value="`*`Value`*`" exp="`*`Exponent`*`"/>`**
Description	Set the *Value* of the system properties referred by the *Path* value. **Note: There are many System Properties which are read only.** The System Properties are the set of data describing a FP Gateway. This includes administrative information like version numbers, licenses etc. which are defined at the creation time of the firmware, as well as information on the hardware configuration and the system state. The system state includes the system time, the system mode the states of the I/O ports etc. The configuration settings defined by the SetConfig are a part of the system state and therefore a part of the system properties. They can therefore also be accessed by the Set command. The difference to the SetConfig command is the way the data is addressed and the structure of the data set. Both commands use a slash separated path to address the data but Set addresses a single value only where SetConfig addresses complex values, for example a complete attribute group. A second difference is in the data itself. All System Properties have a unique address defined by their path. Configurations contain parts which have no unique addresses: For example the PLC "External" could contain several "Devices" on a "Bus" which all have the same tag name "Device". In this case an element can't be addressed uniquely by a path. Therefore, not all elements of the configuration can be addressed by the Set command. Use SetConfig instead.
Parameters	*Path:* Path which addresses the system properties. See Appendix - System Properties for details on system properties. *Value:* Value to set. The syntactical format depends on the value to set. See Appendix - System Properties for details on system properties. Values may be entered directly as decimal, hex, octal or binary values using following special syntax: Example for decimal value "12": value="12" (decimal) value="0xC" (HEX)

value="0o14" (octal)
value="0b1100" (binary)

Set command with HEX format is only supported for unsigned values.

Process and PLC variables may be defined with exponent. In this case, the new value has to be entered relatively to the exponent. Use a dot as decimal point.

Exponent: Exponent of base 10 to specify fix point precision of simpleType = Uint8, Uint16, Uint32, Int8, Int16, Int32 (see chapter 6.5.1).
The variable value will be devided by 10 exp(*Exp*) to get the parameter value.

$\text{valueParameter} = 10^{Exp} \; / \; \text{value variable.}$

The exponent therefore specifies the position of comma within a fix point value. 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

Return

If no error (command is processed): `<Set/>`

On error (command is not processed): see default error frame (chapter 2.4.4)

Example

Set the mode of the Process subsystem:

Client sends:
[<Set _="/Process/Program/Mode" value="Run"/>]

FP Gateway responds
[<Set/>]

Set the the first digital output of the mainboard:

Client sends:
[<Set _="/Process/MB/IO/Q/P0" value="1"/>]

FP Gateway responds:
[<Set/>]

Set a process variable defined with exp="-2":

Client sends:

	`[<Set _="/Process/PV/Variable" value="3.12"/>]`
	
	FP Gateway responds:
	`[<Set/>]`
	
	Set the SignalLED to green (value=1) using an exponent:
	
	Client sends:
	`[<Set _="/Process/MB/SignalLED" exp="1" value="10"/>]`
	
	FP Gateway responds:
	`[<Set/>]`

SetSequence - Set Sequencer profile	
Syntax	See chapter 8.

2.4.6.6 Logging

ReadLog – Read entries from the system log files	
Syntax	**`<ReadLog _="`*`LogFileName`*`" range="`*`entryrange`*`" type="`*`templates`*`" flags="`*`header`*`" fillInterval="`*`interval`*`" maxInterval="`*`tolerance`*`" fillText="`*`string`*`" Viewset="`*`variables`*`" `*`Formats`*`/>`**
Description	The FP Gateway returns the entries from the log file which are in the given range. The range can be composed from entry IDs, time and counts. Some special range commands are available. See chapter 4 on logfiles.
Parameter	*LogFileName:* Name of the logfile to be read. *entryrange:* *all* | *previous n timespan* | *last n timespan* | *start-end* *all* This returns all entries contained inside the given logfile. *last n timespan (exact calculation)* indicates that all entries from the given previous timespan calculated from the current time are to be returned. Where ***n*** is the number of units (must be higher than zero) and ***unit*** the unit itself. Valid ***unit*** values are: *years* indicates that n represents a number of years *months* indicates that n represents a number of months *days* indicates that n represents a number of days *hours* indicates that n represents a number of hours *minutes* indicates that n represents a number of minutes *seconds* indicates that n represents a number of seconds *previous n timespan (smooth calculation)* indicates that all entries from the given previous timespan calculated from the last unit are to be returned. Where ***n*** is the number of units (must be higher than zero) and ***unit*** the unit itself. Valid ***unit*** values are: *years* indicates that n represents a number of years *months* indicates that n represents a number of months *days* indicates that n represents a number of days *hours* indicates that n represents a number of hours

minutes	indicates that n represents a number of minutes
seconds	indicates that n represents a number of seconds

"Previous" will not show values written in the "future", e.g. if you set the clock -1h during daylight saving. During this hour you'll have to use one of the other parameters (e.g. "last") instead.

start-end
Returns all entries contained between the given identifiers. These identifiers - ***start*** and ***end*** - may have one of these formats:

[empty] Means either the first (if ***start***) or the last (if ***end***) entry in the logfile.

#c counts either from start or from end (depends on if used for ***start*** or for ***end***) onto the ***c***-th entry.

Date,Time defines a moment. ***Time*** is in *hh:mm:ss* format (24 hours) and ***Date*** in *YYYY/MM/DD*, and ***Date*** can be omitted if it's about the current day.

ID defines the ID of the entry (if only one ID is given) or entries (if range of IDs is given).

Important:
Keep in mind that when using ***start-end*** you must specify at least ***start*** or ***end*** along with the hyphen. Even if ***start*** or ***end*** is empty, the hyphen must be used.

If ***Time*** range is in the future, data of previous day will be read.

Date has to be used with start AND end, or none of both.

Start Time must be before ***end Time.***

If ***Time*** span reached next day, ***Date*** has to be used.

Last given value will be **end Time** – 1s.

Templates: Predefined logfile formats:

XML: Logfile will be displayed as XML file
CSV*:* "character separated value", e.g. for easy Excel import. (embedded XML frame)
HTML: Logdata will be formatted as HTML table (embedded XML frame)

header:
flags="NoId,NoDate,NoTime,NoNames,NoSec,UseAlias,CRC16,CRC32"
NoId: removes the ID of each entry (only for non-XML structures)
NoDate: removes the date of each entry (only for non-XML structures)
NoTime: removes the Time of each entry (only for non-XML structures)
NoNames: removes the first row with variable names (only for non-XML structures)
NoSec: removes the seconds of the time stamp
UseAlias: adds the variable alias names to the XML logfile output
CRC16: calculates a CRC16 over the logfile output and writes it under the data (only for CSV). See chapter 4.6.

CRC32: calculates a CRC32 over the logfile output and writes it under the data (only for CSV). See chapter 4.6.

interval: Expected log interval. If the time between two log entries exceeds the ***tolerance*** interval, an entry with the content of ***string*** will be added with the timestamp of the last entry + ***interval***.
Can be used to create a fixed log content length if the FP Gateway was switched off or the logging was stopped for a while.

tolerance: Maximum time between two log entries before ***string*** will be added.

string: String added to the log output if ***tolerance*** interval was exceeded (only for CSV format).

variables: List of variables (separated by comma) to be selected for logfile output. The variable names must match the tag names of the record entries.

Formats: See chapter 4.6 for format options like "tabstart", "tabend" etc.

Return

If no error (command is processed), type XML:

```
<ReadLog _="Journal" range="all">
   <ReadLog>
      <LogEntry_ID _="Date,Time">
         <Element _="Logged Data" Name="Alias"/>
            …
      </LogEntry_ID>
   </ReadLog>
```

If no error (command is processed), type CSV:

```
<ReadLog _="Journal" range="all" type="CSV">
   <ReadLog>
   <LogData>
     ID;Date;Time;Element;Element;…;…
     LogEntry_ID;Date;Time;Logged Data;Logged Data;…;…
   </LogData>
   </ReadLog>
```

If no error (command is processed), type HTML:

```
<ReadLog _="Journal" range="all" type="HTML">
  <ReadLog>
  <LogData>
  <table border="1">
    <tr>
      <td>ID</td>
      <td>Date</td>
      <td>Time</td>
      <td>Element</td>
      <td>Element</td>          …
    </tr>
    <tr>
    <td>LogEntry_ID</td>
    <td>Date</td>
    <td>Time</td>
    <td>Logged Data</td>
    <td>Logged Data</td>        …
  </tr>
</table>
</LogData>
</ReadLog>
```

Elements:

LogEntry_ID: ID identifying the logfile entry.

Date: The creation date "YYYY/MM/DD" of the logfile entry.

Time: The creation time "hh:mm:ss" of the logfile entry.

Element: Description of logged element, e.g. variable name of record definition or alias name.

Alias: Alias name of the variable. Only displayed if ***UseAlias*** flag is used for XML outout.

Logged data: Data of logged element, e.g. variable values or event/job results.

On error (command is not processed): see default error frame (chapter 2.4.4)

Example

Get the entries with the IDs 4 – 8.

[<ReadLog _="Journal" range="ID_4-ID_8" ver="y"/>]

Get the entry with the ID 5.

[<ReadLog _="Journal" range="ID_5" ver="y"/>]

Get the entries within a timespan (12:00 to 13:20)

[<ReadLog _="Journal" range="12:00:00-13:20:00"/>]

Get the entries within a timespan on a specific day (12:00 to 13:20 on Dec 24th 2004)

[<ReadLog _="Journal" range="2004/12/24,12:00:00-2004/12/24,13:20:00"/>]

Get last 10 entries.

[<ReadLog> _="Journal" range="#10-"/>]

Getting timespans. Assume that the current time is 12:23

Get entries from last 24h (exact).

[<ReadLog> _="Journal" range="last 24 hours"/>]

This will return all entries from 12:23 previous day to 12:22 current day.

Get entries from last 24h:

[<ReadLog> _="Journal" range="previous 24 hours"/>]

This will return all entries from 12:00 previous day to 11:59 current day.

Clear – Delete content of logfiles	
Syntax	**`<Clear Log="Logfiles"/>`**
Description	Deletes the content of one or several logfiles.
Parameters	*Logfiles:* Logfile or list of logfiles to be deleted. To delete several logfiles with one command, separate the logfile names by comma. Use an asterisk "*" to delete all logfiles.
Return	*If no error (command is processed):* `<Clear/>` *On error (command is not processed):* see default error frame (chapter 2.4.4)
Example	Clear logfiles "JobReport" and "Event". *Client sends:* **`[<Clear Log="JobReport,Event" ver="y"/>]`** *FP Gateway responds:* **`[<Clear/>]`**

2.5 TiXML on SD Card or USB memorystick

FP Gateways with SD Card option (Hx400 and Hx600) can be configured and controlled by TiXML commands saved on a SD Card.

A file with the name "config.txt" must be created on the SD Card, containing following structure:

```
<Batch ResultFile="RESULT.TXT">
   List of TiXML commands
</Batch>
```

The ResultFile attribute defines the file where the TiXML command results will be written to.
The file will automatically be created (if not existing) or overwritten during processing. The file name must be in 8.3 notations. If omitted, RESULT.TXT will be the default name.

The List of TiXML commands may include an unlimited number of commands (see chapter 2.4 for valid commands) and can therefore be used for configuration, controlling or retrieving logged data.

A login to the device may be necessary (see chapter 3.6).

Another option is to use the version attribute:

```
<Batch Version="Version" ResultFile="RESULT.TXT">
    List of TiXML commands
</Batch>
```

The version attribute <Batch version="..."> can be an arbitrary String (<32 characters). This String will be saved in EEProm/Bid after the batch is executed successfully. If the same String occurs again in a batch that is executed later, the batch processing will be skipped.
Content of ResultFile in this case:

```
<Batch deviceID="04242679” date="2016/12/06 10:03:23">
    <Note _="Batch not processed due to equal batchfile-ID"/>
```

```
</Batch>
```

Example (Config.txt to retrieve logged data)

```
<Batch ResultFile="datalog.csv">
    <ReadLog _="Datalogging_0" flags="NoID" type="CSV"
        range="all" ver="v"/>
</Batch>
```

Config.txt to configure address book. The Signal-LED will be switched on/off to indicate the user when the processing of the file is finished. During configuration, the FP Gateway will be set to stop mode:

```
<Batch>
<Login/>
    <Set _="/Process/Program/Mode" value="Stop"
       ver="y"/>
    <Set _="/Process/MB/SignalLED" value="1" ver="y"/>

    <SetConfig _="TEMPLATE" ver="y">
        <AddressBook>
            <MySelf>
              <Email _="tixi-gateway@tixi.com"/>
            </MySelf>
            <Contact_0>
              <Email _="tixi-support@inovolabs.com"/>
            </Contact_0>
        </AddressBook>
    </SetConfig>

    <Set _="/Process/MB/SignalLED" value="0" ver="y"/>
    <Set _="/Process/Program/Mode" value="Run"
       ver="y"/>
</Batch>
```

3. Main XML Databases

This chapter describes the different databases used by a basic project creation.

3.1 Introduction

We recommend the use of "TICO – TiXML Console" to easily create databases out of the included template library.

If you use any terminal or self written program you need to stop the job processing before sending the first SetConfig by sending the following command:

```
[<Set _="/Process/Program/Mode" value="Stop" ver="y"/>]
```

After uploading the project/database you need to start the job processing (this is automatically done by FP Gateway reset):

```
[<Set _="/Process/Program/Mode" value="Run" ver="y"/>]
```

3.1.1 References

The great advantage of XML databases is the possibility of linking (cross reference) the content and save configuration time. For example, instead of configuring the same fax number in each of the message job templates, you only have to make a reference to the fax number in the address book. If you change the number in the address book, it’s automatically changed for all related message job templates.

A reference to a value is introduced by the reference symbol ® (written as XML entity `&#xae;`) followed by the path to the value. The path has to be ended by a semicolon if text is following.

Depending on the database location, you have to use different reference paths.

Examples

- Reference to a parameter received by EventState, DoOn or incoming message:
  `&#xae;~/parameter;`
- Reference within same database
  `&#xae;/D/Group/entry;`

  Example
  Reference inside the TEMPLATE database, e.g. from MessageJobTemplate to AddressBook:

  ```
  &#xae;/D/AddressBook/Contact_0;
  ```

  or MessageText to another MessageText:

  ```
  &#xae;/D/UserTemplates/LocationText;
  ```

- Reference accross databases
  `&#xae;/DATABASE/Group/entry;`

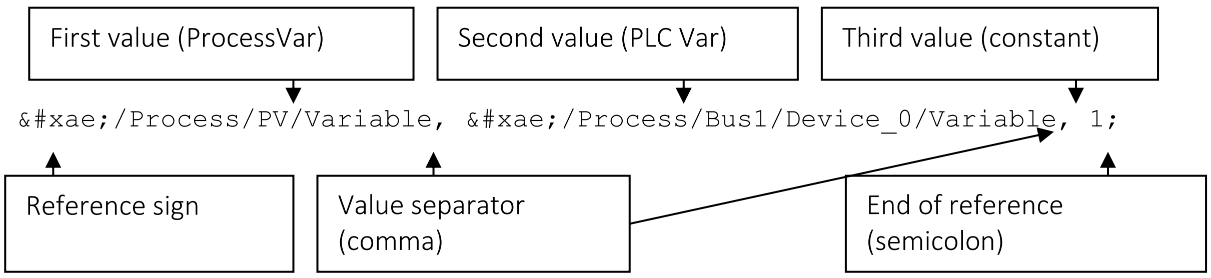
  Example
  Reference from the TEMPLATE database to USER database:

```
&#xae;/USER/Location/PhoneNumber;
```

- Path to a variable, e.g. from EventHandler Set command or within process variable:
  `/Process/Bus1/Device_0/Variable_0` (without reference symbol)

- Project-Structure related Path from EventHandler to a MessageJobTemplate:
  `MessageJobTemplates/Alarm_0` (without reference symbol)

Alternative reference:
If a reference can not be resolved by the job processor, the job will be canceled with an error log entry. Therefore it is sometimes useful to cascade references with alternative values, which are separated by comma, e.g.:



```
&#xae;/Process/PV/Variable, &#xae;/Process/Bus1/Device_0/Variable, 1;
```

Alternative values are only possible for references with `"&#xae;"` but not for paths (e.g. EventHandler `"set"` path, EventHandler "sendmail" path) instructions.

3.1.2 Time parameters

Every time value without unit will be interpreted as "milliseconds".
A unit can be added to every value to use larger time periods:
"s" for seconds, "m" for minutes, "h" for hours, "d" for days.

Example

`<Delay _="500"/>`	500ms delay
`<Delay _="30s"/>`	30s delay

3.2 Dialling Properties of the Location

The properties, which describe the telephone connection to which the device is attached, are called "Location". This is because these properties depend on the place where the FP Gateway is installed.

Database path: /USER/Location

```
<Location>
   <CountryPrefix _="00"/>
   <CountryCode _="49"/>
   <AreaPrefix _="0"/>
   <AreaCode _="30"/>
   <LocalDialPrefix _=""/>
   <LongDialPrefix _=""/>
   <PhoneNumber _="12345678"/>
   <InternalDialPrefix _=""/>
   <ExtensionNumber _=""/>
   <DialRules _="Tone,NoWaitForDialTone"/>
```

Insert your own data

```
    <NumberFormat _="*"/>
</Location>
```

Name	*Description*
CountryPrefix	Country prefix for international calls, e.g. 00 inside Germany.
CountryCode	Country code of location without CountryPrefix, e.g. 49 for Germany
AreaPrefix	Area prefix for domestic long distance calls, e.g. 0 for Germany, 1 for the U.S.A.
AreaCode	Area code of location without AreaPrefix, e.g. 30 for Berlin GSM-Note: If you are using a FP Gateway GSM please enter the GSM network code of your GSM provider, e.g. German T-Mobile: 171 German Vodafone: 172 German Eplus: 177 etc.
LocalDialPrefix	PBX prefix to get an outside line for local calls (to dial numbers with same area code). May be left blank.
LongDialPrefix	PBX prefix to get an outside line for long distance calls (to dial numbers with different area code). May be left blank.
PhoneNumber	Local phone number of the location without any prefix or area code. If *no PBX* is used this is the complete phone number (e.g. 123456500). If an *PBX is used,* this is the phone number of the PBX. In this case, the complete phone number will be created by this field and the ExtensionNumber.
InternalDialPrefix	PBX prefix to receive a dial tone for internal calls. May be left blank.
ExtensionNumber	Last digits of the phone number defined by the PBX extension.
DialRules	Defines the FP Gateway dial method. *DialMode:DialToneRecognition* *DialMode (used for PSTN devices only):* Tone....Tone dialling Pulse....Pulse dialling *DialToneRecognition:* WaitForDialTone....Wait for a dial tone. NoWaitForDialTone....Do not wait for a dial tone. e.g.: Use tone dialling and do not wait for dial tone: `Tone,NoWaitForDialTone`
NumberFormat	Specifies how the device will dial the recipients number * Network related (default) If it's an FP GSM device, all numbers are dialed canonical. If it's a PSTN/ISDN device, all numbers are dialed depending on the location settings. n location related (PSTN only) All numbers are dialed depending on the location settings (see chapter 3.2.1). c canonical (GSM only) All numbers are dialed canonical (e.g. +49172123456).

During development, define this location for the place where you will test the FP Gateway. Later at the place where the FP Gateway is installed, change the location settings for this place. Due to the use of international phone numbers you don't need to change the phone numbers in the ISP's data or in the address book to add or remove dial-prefixes etc. Simply change the Location and FP Gateway dials the proper number.

Defining a bad location is a most common error in configuring FP Gateway. Please follow the following hints provided.

First check whether you are using a telephone extension or not.

If no PBX is used the configuration is very simple:

1. Select the template corresponding to the country. In this template all settings for the country should be predefined (`CountryCode, AreaCode, CountryPrefix, AreaPrefix`).
2. Set all `DialPrefix` fields as blank entries "".
3. Also leave the `ExtensionNumber` blank.
4. Insert the `PhoneNumber`.
5. Insert the `DialRules`: typically `Tone,NoWaitForDialTone`.

If a PBX is used the configuration depends on the properties of the PBX.

1. Select the template corresponding to the country. In this template all settings for the country should be predefined (`CountryCode, AreaCode, CountryPrefix, AreaPrefix`).
2. The PBX may require some prefixes you have to dial to get an outside line. Furthermore, some PBX require different prefixes for different call types; others require the same or nothing for the call types. Our location defines three call types:
   a). Internal call (`InternalDialPrefix`): the call resides inside the PBX.
   b). Local call (`LocalDialPrefix`): the call has the same area code as the location.
   c). Long distance Call (`LongDialPrefix`): the call goes outside the area code.

   When the same prefix is used for different call types, insert the prefix in each prefix field. When no prefix is used for a call type, leave it blank.
3. PBX typically define a range of internal numbers, which can be called to get a person inside the PBX, e.g. from the number 123456-200 you can dial 300 instead of 123456-300. In this example the last three digits are called Extension Number and the first five digits are the phone number (the same for all extensions inside the PBX).

   If your extension defines internal (short) numbers, fill both fields `PhoneNumber` and `ExtensionNumber` with the right numbers.
4. Insert the Dial Rules: typically `Tone,NoWaitForDialTone`.

   The value Pulse for pulse dialling method is used by old PBX only. But check whether your PBX supports dial tone recognition or not.

3.2.1 How the Device dials

There are several events where the FP Gateway has to dial a phone number, e.g. for internet connections, fax or SMS.

All these recipient numbers have to be inserted in canonical (international) format like

```
+CountryCode-AreaCode-PhoneNumber
```

Referring to the location details, the device checks if the recipients number is

- in the same country
- in the same area
- within the same PBX

and therefore dials only the necessary part of the number.

Example

1. The device location is set to

```
<Location>
   <CountryPrefix _="00"/>
   <CountryCode _="49"/>
   <AreaPrefix _="0"/>
   <AreaCode _="30"/>
   <LocalDialPrefix _=""/>
   <LongDialPrefix _=""/>
   <PhoneNumber _="12345678"/>
   <InternalDialPrefix _=""/>
   <ExtensionNumber _=""/>
   <DialRules _="Tone,NoWaitForDialTone"/>
</Location>
```

a) If the recipient is "+49-30-5555555" the modem will only dial "5555555" (phone number) because the CountryCode and AreaCode are the same.

b) If the recipient is "+49-40-4444444" the modem will dial "0 40 4444444" (area prefix, area code and phone number) because the CountryCode is the same but the AreaCode is different.

c) If the recipient is "+44-170-3333333" the modem will dial "00 44 170 3333333" (country prefix, country code, area code and phone number) because all settings are different.

2. The device location is set to

```
<Location>
   <CountryPrefix _="00"/>
   <CountryCode _="49"/>
   <AreaPrefix _="0"/>
   <AreaCode _="30"/>
   <LocalDialPrefix _="9"/>
   <LongDialPrefix _="1"/>
   <PhoneNumber _="12345678"/>
   <InternalDialPrefix _=""/>
   <ExtensionNumber _=""/>
   <DialRules _="Tone,NoWaitForDialTone"/>
</Location>
```

a) If the recipient is "+49-30-5555555" the device will only dial "9 5555555" (local dial prefix and phone number) because the CountryCode and AreaCode are the same.

b) If the recipient is "+49-40-4444444" the device will dial "1 0 40 4444444" (long dial prefix, area prefix, area code and phone number) because the CountryCode is the same but the AreaCode is different.

c) If the recipient is "+44-170-3333333" the device will dial "1 00 44 170 3333333" (long dial prefix, country prefix, country code, area code and phone number) because all settings are different.

3. The device location is set to

```
<Location>
   <CountryPrefix _="00"/>
   <CountryCode _="49"/>
   <AreaPrefix _="0"/>
   <AreaCode _="30"/>
   <LocalDialPrefix _=""/>
   <LongDialPrefix _=""/>
   <PhoneNumber _="123456"/>
   <InternalDialPrefix _="!"/>
   <ExtensionNumber _="789"/>
   <DialRules _="Tone,NoWaitForDialTone"/>
</Location>
```

a) If the recipient is "+49-30-123456-111" the device will only dial "! 111" (internal dial prefix !=FLASH, and extension number) because the CountryCode, AreaCode and phone number are the same.

3.3 Device's User Data

The FP Gateway has a database called "USER" which contains some modem related settings. This includes the settings for GSM, redial and for call acceptance.

Database path: /USER/USER

```
<USER>
   <Handshake _="RTSCTS"/>
   <InitString0 _="ATX3M1L1"/>
   <ModemProtocol _="default"/>
   <IsdnDataChannelID _="*"/>
   <IsdnFaxChannelID _="*"/>
   <DChannelProtocol _="DSS1"/>
   <ModemListenMode _="OFF"/>
   <ModemParams _=""/>
   <BoxName _="FP Gateway-ID"/>
   <BoxNumber _="+49-30-1234567"/>
   <TimeZone _="+0100"/>
   <MaxDialAttempts _="3"/>
   <MemForInMails _="0"/>
   <RedialDelay _="60s"/>
   <RingCounter _="0"/>
   <LogInComCalls _="1"/>
   <AccountQuery _="*100#"/>
   <AccountExpiry _="*101*1#"/>
   <AccountResponse _="amount:Guthaben:;valid:am;format:dd.mm.yyyy"/>
   <Pin1 _="1234"/>
   <GPRS _="Off"/>
</USER>
```

Insert your own data

Name	*Description*
Handshake	COM-Port Handshake for TiXML communication. **RTSCTS** Hardware Handshake (default) **XONXOFF** Software Handshake
BoxName	Name of the FP Gateway when it is used for sending in Fax message headlines. The BoxName is also used as the hostname for DHCP entries.

BoxNumber	Canonical phone number of the FP Gateway. It identifies the phone network connection of the FP Gateway when it is used for Fax message headlines. Syntax: ***CountryIDAreaCodeLocalPhoneNumber***
TimeZone	Time zone where the FP Gateway is located. The value is the difference in hours and minutes from GMT. Syntax: ***+/-HHMM***
MaxDialAttempts	Maximum number of dial attempts **1...10**. **1** is recommended as redial should rather be configured by the SendMail command. See chapter 3.7.1.
RedialDelay	Time to wait between dial attempts in seconds **30s...255s** The timer starts after the failed sendmail.
IsdnDataChannelID	Multiple Subscriber Number (MSN) for ISDN data calls. ***** answers on all numbers (default). **nn** MSN of the device (up to 16 digits)
RingCounter	Number of rings until the FP Gateway answers an incoming call. This doesn't affect SMS receipt. **0** Ignore all incoming calls **1....10** ring counter.
LogInComCalls	Enables the logging of all incoming calls into the "IncomingMessage" Logfile. **0** disable logging calls **1** enable logging calls
AccountQuery	String to query the SIM card credit Germany: e.g. *100# (D1, D2, O2, Eplus) empty: deactivated Outside germany `AccountResponse` may be required.
AccountExpiry	String to query the SIM card expiration date, not necessary if identical to AccountQuery. Germany: e.g. *100# (D2, Eplus), *101*1# (D1), *102# (O2) empty: deactivated `AccountResponse` required.
AccountResponse	Response parser for AccountQuery and AccountExpiry. Format: `"amount:[word before credit];valid:[word before expiry];format:[expiry format]"`
Pin1	The PIN for the SIM card.
Pin2	A second PIN for the SIM card, if required.
GPRS	Switches between GSM-CSD and GPRS functionality. Off: GPRS deactivated (default) On: GPRS activated **Note:** If GPRS is activated, only TCP/IP communication is possible (email, HTTP, TFTP, TiXML). Services for sending SMS, Fax, Pager and dial in for TransMode or remote configuration are inaccessible.
InitString0	Should normally not be altered. Setting can be used to force the mobile modem to use a specific carrier. Useful if you don't want to login into different carrier to prevent roaming expenses. Example: `<InitString0 _="AT+COPS=1,0,`**`26201`**`"/>` whereas **`26201`** is the world wide valid operator code of your choice (3 digit MCC followed by 2 digit MNC; other parameters 1,0 are fixed !) see also: https://en.wikipedia.org/wiki/Mobile_country_code

Note:
The GSM PIN is not deleted by a factory reset! To delete the GSM PIN it is necessary to overwrite it with an empty value.

3.4 Address Book

Each message created by the FP Gateway must include a sender and one or more recipient addresses. The FP Gateway provides an address book for all recipients and senders of messages. So if you create different messages which are sent to the same recipjent you can use the same contact in the message job templates. This is a simple way to manage your addresses and reduce errors on configuration.

We recommend using not more than 100 addressbook entries, as otherwise the processing speed may drop below reasonable levels.

The address book is stored in the 'TEMPLATE' database. Each contact can contain addresses for different message types (SMTP, SMS, TextFax etc).

Database path: /TEMPLATE/AddressBook

```
<AddressBook>
   <MySelf>
      <Email _="user@domain.com"/>
      <SMS_No _="+49-30-1234567"/>
      <Fax _="+49-30-1234567"/>
   </MySelf>
   <Contact_0>
      <Email _="office@domain2.com"/>
      <SMS_No _="+49-174-1234567"/>
      <SMS_Provider _="D2"/>
      <Fax _="+49-30-7654321"/>
      <CityRuf _="3949000"/>
      <Pager_Provider _="CityRuf"/>
      <URL _="http://www.devicecontrolnet.com/notification/">
      <URLPort _="80"/>
      <User _="WebAccess"/>
      <Password _="WebAccessPwd"/>
   </Contact_0>
</AddressBook>
```

Contacts

Contact	
Syntax	***<ContactName>*** ***List of Address Entries*** ***</ContactName>***
Description	Attribute group which defines a symbolic address as a contact. The symbolic address can be used as sender and recipient of messages.
Elements	*ContactName:* Name of the contact. This is the symbolic address inserted in the message job templates. It must be unique in the address book. *List of Address Entries:* List of attributes defining the addresses of the contact for the different message types.

Example Contact 'MySelf' used as sender for the messages. 'MySelf' is the symbolic address. An internet address, a SMS address a Fax address are defined.

```
<MySelf>
   <Email _="tam@domain.com"/>
   <SMS_No _="+49-30-1234567"/>
   <Fax _="+49-30-1234567"/>
</MySelf>
```

There are several address entries. These entries correspond to a certain message type (SMTP, FAX, SMS, Express-E-Mail, CityRuf, HTTP notification). You can insert one address for each transport type within one contact.

```
<Contact_0>
   <Email _="office@domain.com"/>
   <SMS_No _="+49-174-1234567"/>
   <SMS_Provider _="D2"/>
   <Fax _="+49-30-7654321"/>
   <URL _="http://www.devicecontrolnet.com/notification/">
   <URLPort _="80"/>
   <User _="WebAccess"/>
   <Password _="WebAccessPwd"/>
</Contact_0>
```

Address entry

Name	*Description*
Email	Internet address of the contact (e.g. `tam@domain.com`). If an AddressBook entry contains more than one email entry, the email will be sent to all receivers.
SMS_No	SMS telephone number of the contact (e.g. +49-161-1234567). May be entered in short number format (e.g. "8888") or canonical format, if the `NumberFormat` in the SMS-Provider is configured (see chapter 3.10).
SMS_Provider	Name of the SMS provider used when the contact is a SMS recipient. In this case the related SMS dial in number of the provider is used. See chapter 3.10
Fax	Fax number of the contact (for example +49-30-1234567) written in canonical format: *+CountryCode-AreaCode-LocalPhoneNumber*
Express-Email (obsolete)	Express-E-Mail address of the contact (for example TAM+49-30-1234567). This is inserted into the header of Express-E-Mails. It consists of the user name and the international phone number of the receiving FP Gateway. The canonical phone number format is: *+CountryCode-AreaCode-LocalPhoneNumber*
CityRuf	CityRuf number of the contact, for example 3949000.
Pager_Provider	Name of the pager provider used when the contact is a pager recipient. In this case, the related pager dial in number of the provider is used. See chapter 3.10
URL	URL of the HTTP server that receives and processes upcoming alarms. A slash "/" must be at the end of the URL, if a folder ist requested.
URLPort	TCP/IP port of the HTTP server.
User	Username to get access to the webserver specified by URL.
Password	Password to get access to the webserver specified by URL.

For each recipient of your messages you have to prepare the addresses of the message types by which you want to send messages to him.

3.5 Internet Access (ISP)

The FP Gateway provides the ability to send emails via the Internet. FP IoT Gateway does this via its LAN connection and FP Gateway calls a dial-in node of an Internet Service Provider (ISP) and establishes a TCP/IP connection to the Internet. This TCP/IP connection is embedded into a Point to Point Protocol (PPP) connection which is established between FP Gateway and the dial-in node of the ISP. FP Gateway GSM may also use a GPRS connection to communicate with the internet.
When the TCP/IP connection is ready, the FP Gateway uses the SMTP server of the provider to send email. The related ISP data has to be configured in the 'ISP' database in the 'ISP' section.

Database path: /ISP/ISP

```
<ISP>
   <PPPComm>
      <PPPUserName _="user"/>
      <PPPPassword _="pass"/>
      <AuthentFlags _="3"/>
      <FirstDNSAddr _="194.25.2.129"/>
      <SecondDNSAddr _="193.158.131.19"/>
      <KeepConnected _="24h">
      <EchoInterval _="1m"/>
      <EchoTimeout _="30s"/>
      <EchoTarget _="www.google.de"/>
      <OnTCPError _="KeepConnected">
   </PPPComm>
   <SMTP>
      <Flags _="ESMTP"/>
      <mailserver_name _="domain.com"/>
      <mailserver_ip _="192.168.0.1"/>
      <SSL _="SSLMode" />
      <Port _="25"/>
      <Username _="user"/>
      <Password _="pass"/>
      <ownhost_ip _="[&#xae;/Ethernet/AssignedIP;]"/>
   </SMTP>
   <POP3>
      <mailserver_name _="domain.com"/>
      <SSL _="SSLMode" />
      <Port _="110"/>
      <Username _="user"/>
      <Password _="pass"/>
      <Flags _="DontDelete"/>
      <Filter _="string"/>
      <Lines _="50"/>
   </POP3>
   <Modem>
      <RemotePhoneNumber _="+49-30-1234567"/>
      <MediaType _="DATA"/>
      <ModemProtocol _="syncPPP"/>
   </Modem>
   <GPRS>
      <APN _="web.vodafone.de"/>
   </GPRS>
```

Insert your own data

```
</ISP>
```

Name	*Description*
PPPUserName	User name of the PPP log-in (provided by the ISP). *Please note*: When using the GSM socket modem “Sierra Wireless Q2687RD” you need to put at least one arbitrary character into this field, even if it is not required by the mobile service provider.
PPPPassword	Password of the PPP log-in (provided by the ISP). *Please note*: When using the GSM socket modem “Sierra Wireless Q2687RD” you need to put at least one arbitrary character into this field, even if it is not required by the mobile service provider.
AuthentFlags	PPP authentication method: 1 PAP (plain text) only 2 CHAP (challenge handshake) only 3 auto
FirstDNSAddr	IP of DNS #1 (provided by your ISP, omit if dynamic)
SecondDNSAddr	IP of DNS #2 (provided by your ISP, omit if dynamic)
KeepConnected	Time the FP Gateway will stay online after successfully completing the last message job during the connection. Useful to get a permanent GPRS connection initiated by "`Connect`" EventHandler (see chapter 3.7.1).
EchoInterval	Interval in which the device will ping the EchoTarget to check the IP communication.
EchoTimeout	Timout after which the IP communication check fails.
EchoTarget	ICMP host used by the IP communication check. Can be specified as FQDN or IP address.
OnTCPError	Controls the behaviour on TCP errors, if connection is kept online using "KeepConnected". Disconnect: Reestablish connection on TCP errors KeepConnected: Ignore TCP errors
SMTP/Flags	Enter value "`POPBeforeSMTP`" if you need POP3-before-SMTP authentication. Enter value "`ESMTP`" if you need SMTP authentication.
mailserver_name	Domain name of senders email address or name or IP address of POP3/SMTP server.
mailserver_ip	Name or IP address of POP3/SMTP server.
SSL	SSLMode: <empty>, NONE don’t use SSL STARTTLS use STARTTLS DIRECTSSL direct SSL-connection
Port	TCP port of the SMTP/POP3 server. Defaults: POP3: 110 SMTP: 25
SMTP/Username	Only for ESMTP: User name for the ESMTP server (provided by the ISP)
SMTP/Password	Only for ESMTP: Password of the ESMTP server (provided by the ISP)
ownhost_ip	Hostname or IP of the the FP Gateway, used by HELO command
POP3/Flags	Enter value "`DontDelete`" to prevent mails from being deleted after successful pickup.
POP3/Username	User name for the POP3 server (provided by the ISP)
POP3/Password	Password of the POP3 server (provided by the ISP)

Filter	Filter word to be found within the collected email, otherwise the device will ignore the email.
Lines	Number of lines the device will search for the filter word.
RemotePhoneNumber	International phone number to call the ISP's dial-in node or code for GPRS connection. The format of the international phone number is: *+CountryCode-AreaCode-LocalPhoneNumber* **GSM Note:** Most ISP offer a different phone number for calls from the GSM network. Please contact your ISP to get the GSM number if you are using a FP Gateway GSM. Common GPRS dialup code: `*99***1#`
ModemProtocol	ISDN/GSM-Protocol used to connect to the ISP. Values: "default" (uses V.32 or X.75-NL) "X.75-NL" "syncPPP" "V.120" "V.110" (for GSM) "X.75-T.70" "ML-PPP" "HDLC-Transp" "BYTE-Transp"
GPRS/APN	Access Point Name of the GPRS backbone.

Note:
For FP Gateways the ISP groups "`PPPCom`" and "`Modem`" must be created (may be empty) for historical reasons.

3.5.1 CloudConnector

For information about the cloudConnector, consult the "Tixi cloudConnector Introduction and Quick Start Guide".

3.6 Access rights

You can protect the access to the FP Gateway against unauthorized access. The factory configuration of the FP Gateway has no protection activated, so if you forget the password, you can always gain access to FP Gateway by using the hardware factory reset.

There are three levels of protection:

- **no protection** (no Login is required, **factory default**)
- **password** protection (a password is required for login, user name is empty)
- **user aware** protection (a user name and a password are required for login)

Login with CallerID verification is only available for remote switching (see chapter 9.5).

Within the AccRights group it is possible to define access groups with different access types (services) and assign users to these groups. The password may be encrypted (Base64+ThreeWay or Keyed-MD5).

The "AccRights" are part of the USER database:

Database path: /USER/AccRights

Access Rights	

Syntax	
	```
	<AccRights>
	      <Groups>
	            **<*Groupname*>**
	               **<*Service* AccLevel="*Level*"/>**
	            **</*Groupname*>**
	      </Groups>

	      <User>
	           **<*Username* Plain="*PlainPwd*" Group="*UserGroup*" />**
	           **<*OA_nnn*   Plain="*PlainPwd*" Group="*UserGroup*" />**
	           **<*Username* Pwd="*Pwd*"  Group="*UserGroup*"**
	              **Callback="*number*"/>**
	           **<*Def_Service* Pwd="*Pwd*" Group="*UserGroup*"/>**
	      </User>
	<AccRights>
	```

Description	Configuration of access rights. Each user is assigned to an access group. The access group specifies the accessible services and access levels. As soon as a "username" is defined, all services are locked. To unlock services for everyone, a "Def_" user without password has to be defined for these services. A user also gets access to all services that are not expicitly denied within his group. To prevent this, these services has to be locked by AccLevel="-1". If a user is member of two groups and a service is disallowed in his first group but allowed or not specified in his secound group, he will get access. Disallowed services have to be blocked in all assigned groups. For TSAdapter access rights, at least an user ADMIN has to be configured.
Elements	*Groupname:* Name of an access group. Serveral access groups with different services may be defined. *Service:* Service that may be accessed by this group: LocalLogin local access via serial port RemoteLogin access via dialin EthernetLogin access via TCP/IP (TiXML) CardLogin access via SD Card Message access via incoming message WebServer access to the webserver TFTP access via TFTP

TSAdapter	access via TS-Adapter (only Hx71/Hx76)

Level:
Access Level of this group for the specified service.

-1	access protection disabled
1	access protection enabled
>1	group access level

Username:
Name of an user with access rights or email alias (see chapter 9.5 for more details).

OA_nnn:
CallerID or email address used to secure remote switching (see chapter 9.5 for more details).

PlainPwd:
Password assigned to an user (plain text). Maximum 79 characters, for service "message" maximum 25 characters.

Pwd:
Password assigned to an user (encrypted, Base64+ThreeWay or Keyed-MD5). Maximum 59 characters.

UserGroup:
List of groups (see groupname) the user will have access to (separate by comma).

Number:
Callback number for TSAdapter service (only devices with MPI interface)

Def_Service:
Default user for each service. Replace "Service" by service name. Default user will be used if login username is unknown or empty.

Example

Three groups are defined: Group "login" is used for configuration access to the device, group "RemoteControl" is used for processing incoming messages, group "Step7" is used for Siemens S7-300/400 TeleService access.

User Tom is member of group "RemoteControl", therefore he can send messages to the FP Gateway but he cannot login to the device.
Remote switching with the password "TIXI" is only valid from a mobile phone with the number +491721234567.

User Paul is member of group "Login" and "RemoteControl", therefore he has full access to all services.
The technicians "Martin" and "Daniel" are not defined but they may use the passwords "Winter" for remote control and "Summer" for local login (default access).
User ADMIN is able to access a connected S7-300/400 PLC via callback using the Step7 TeleService software.

```
<AccRights>
      <Groups>
            <Login>
               <LocalLogin AccLevel="1"/>
```

```
              <RemoteLogin AccLevel="1"/>
              <EthernetLogin AccLevel="1"/>
              <CardLogin AccLevel="1"/>
              <Message AccLevel="-1"/>
              <WebServer AccLevel="-1"/>
              <TFTP AccLevel="-1"/>
              <TSAdapter _="-1"/>
           </Login>
           <RemoteControl>
              <Message AccLevel="1"/>
              <WebServer AccLevel="10"/>
           </RemoteControl>
           <Step7>
              <TSAdapter _="1"/>
           </Step7>
     </ Groups>
     <User>
       <Tom Plain="Spring" Group="RemoteControl" />
       <OA_491721234567 Plain="TIXI" Group="RemoteControl"
     />
       <Paul Pwd="Agshezg435G73gg723=="
Group="Login,RemoteControl" />
       <Def_RemoteLogin Plain="Winter"
     Group="RemoteControl"/>
       <Def_LocalLogin Plain="Summer" Group="Login"/>
       <Def_CardLogin Plain="Summer" Group="Login"/>
       <ADMIN Plain="Autumn" Group="Step7"
     Callback="+491721234567"/>
     </User>
<AccRights>
```

3.7 Event Handler

The general instructions are defined in the '**EVENTS**' database. The content of this database configures the event handler of the FP Gateway. The database contains some attribute groups. Each group is named by an event and contains processing instructions as attributes. These instructions are processed by the event handler from top to bottom.

We recommend using not more than 100 event handlers, as otherwise the processing speed may drop below reasonable levels.

The following example shows the configuration for two events '`Alarm_0`' and '`Alarm_1`', each has its own commands:

Database path: EVENTS/EventHandler

```
<EventHandler>

   <Alarm_0>
      <SendMail _="MessageJobTemplates/Alarm_0"/>
      <SendMail _="MessageJobTemplates/Alarm_1"/>
   </Alarm_0>

   <Alarm_1>
```

Event Handler Group

Event Handler Commands

```
        <SendMail _="MessageJobTemplates/Alarm_1"/>
     </Alarm_1>

  </EventHandler>
```

Event Handler Configuration	
Syntax	***<EventName>*** ***CommandList*** ***</EventName>***
Description	Attribute group which defines the instructions to be processed each time the event handler is triggered by a DoOn command or an EventState (see chapter 6.2).
Elements	*EventName:* Name of the event triggered by the DoOn command or by the process subsystem (see chapter 6.3) on the event parameters). *CommandList*: List of Attributes describing commands for the event handler which are processed from top to bottom (see next chapter).

Example	Event handler configuration for the 'Alarm_0' event. It lets the event handler send an alarm message defined by the message job templates 'Alarm_0' and set an output port.
	
	`<Alarm_0>`
	`    <SendMail _="MessageJobTemplates/Alarm_0"/>`
	`    <Set _="/Process/MB/IO/Q/P0" value="1"/>`
	`</Alarm_0>`

3.7.1 Commands

SendMail Command	
Syntax	**<SendMail _="*Template*">** ** <OnOK _="*OnOKEvent*"/>** ** <OnError _="*OnErrorEvent*"/>** ** <MaxRepeat _="*MaxRepetitions*"/>** ** <Interval _="*IntervalTime*"/>** ** <ConfirmID _="*ID*"/>** ** <Timeout _="*Timeout*"/>** ** <OnTimeout _="*OnTimeoutEvent*"/>** ** <Delay _="*DelayTime*"/>** ** <Condition _="*Variable*"/>** ** <Priority _="*Priority*"/>** ** <KeepConnected _="*OnlineTime*"/>** **</SendMail>**
Description	Event handler command. It lets the event handler send a message using the given message job template (see chapter 3.9).
Elements	*Template* Name of the message job template which is used to generate the message job for this event. *OnOKEvent*

	Name of the event to be triggered when the sending of the ***Template*** message job didn't fail or a confirmation message was received. If empty or omitted, nothing happens in that case. *OnErrorEvent* Name of the event to be triggered when the sending of the ***Template*** message job failed. If empty or omitted, nothing happens in that case. *MaxRepetitions* Determines how many attempts are made to execute the ***Template*** message job before *OnOK, OnError or OnTimeout* will be triggered. (Requires *Interval* time) *IntervalTime* Determines the delay between the attempts of ***MaxRepetitions***. **Default** is 30s. Timer starts after failed sendmail. *ID* The ID (0-65533) is used as identification for the message if a confirmation is requested. See next page for details about message confirmation. *Timeout* Determines after which time the OnTimeout event is invoked. Timer starts with begin of sendmail. *OnTimeoutEvent* Name of the event to be triggered when the message was successfully sent but no confirmation was received after the time determined in ***Timeout***. If empty or omitted, nothing happens in that case. *DelayTime* Delays the sending of the messsage for the given time. Timer starts with created message. *Variable* Condition for message dispatch. If the bit variable value specified by the path is 1 (TRUE), the message will be sent immediately. If the value is 0 (FALSE) the sending of the message will be delayed according to ***DelayTime***. Supported variable types: External variables, process variables, elements of the "/Process/MB/" tree excluding its subfolders. Not supported variable types: database entries, FP Gateway I/Os. The condition can be checked again by using the EventHandler command `CheckJobConditions` (see below). *Priority* Sets a priority for the alarm. If several alarms are activated at the same time, the device will send out the alarm with the highest priority (255 = highest) at first. Possible priorities: 1-255 *OnlineTime* Defines how long the device will keep the PPP connection established, after complete sending of the email. (SMTP only)
Example	Event handler configuration for the '`Alarm_0`' event. It lets the event handler send a message with priority 2 using the template '`Alarm_0`'. If sending of the message fails two times, the alarm will be retriggered (loop). If it was sent successful, the FP

Gateway will wait 10 minutes for a confirmation message. If the confirmation is missing, the alarm will be retriggered (loop).

```
<Alarm_0>
  <SendMail _="MessageJobTemplates/Alarm_0">
    <OnError _="Alarm_0"/>
    <MaxRepeat _="1"/>
    <Interval _="180s"/>
    <ConfirmID _="1"/>
    <Timeout _="10m"/>
    <OnTimeout _="Alarm_0"/>
    <Priority _="2"/>
  </SendMail>
</Alarm_0>
```

Alarm cascading with OnOK, OnError, OnTimeout

The three cascading commands can be used seperately or combined.

- `OnOK` will be triggered if the message was sent successfully.
- `OnError` will be triggered if the message failed to create or transmit.
- `OnTimeout` will be triggered if the message was not acknowledged within timeout time.

Note:
- `OnOK` combined with `OnTimeout` will be triggered if the message has been acknowledged.
- `OnTimeout` will not be triggered if the message failed to create or transmit; therefore we recommend combining `OnTimeout` with `OnError`.
- If the `OnTimeout` timeout is shorter than the time for all `MaxRepeat intervals`, `OnTimeout` may be triggered even if the message was not sent successfully.

Confirmation of messages

A successfully sent message is not always a guarantee that the message has reached the recipient. To check this, it is possible to request a confirmation from the recipient. The confirmation request is activated with the line **`<ConfirmID _="`*`ID`*`"/>`** together with the **`OnTimeout`** attribute. The *ID* (0-65533) is an identifier for the sent message. The ConfirmID has to be included in the message in form of the "`_Fingerprint`" variable.

In this message text template, the fingerprint is included in the subject of the message.

Database path: /TEMPLATE/UserTemplates

```
<Message_0>
   <Subject>
      <C _="Alarm! Confirmation needed: &#xae;~/_Fingerprint; "/>
   </Subject>
</Message_0>
```

An event handler using this message may have the following form:

Database path: /EVENTS/EventHandler

```
<Alarm_0>
    <SendMail _="MessageJobTemplates/Alarm_0">
      <Interval _="120s"/>
      <MaxRepeat _="2"/>
      <Timeout _="180s"/>
      <OnError _="ErrorLog"/>
      <OnTimeout _="TimeoutLog"/>
      <OnOK _="OKLog"/>
      <ConfirmID _="100"/>
    </SendMail>
</Alarm_0>
```

The message would look similar to this:

Alarm! Confirmation needed: CID2VeFhc7SyfaJMT/h

The confirmation is performed by the `Confirm` command of an event handler. There are two ways to invoke this event handler:

1. As a reply/forwarded message with the received subject included. The FP Gateway searches the text of received SMS and the subject lines of Email messages for a fingerprint. In this fingerprint the confirmation ID, the date and the time is encrypted. So it is not possible for an unauthorized person to fake a confirmation. Furthermore, since every fingerprint is unique, it is not possible to use a received fingerprint twice for confirmation. If a fingerprint was received by the FP Gateway, it invokes a special system event `/System/Confirmation` (see chapter 3.7.3) which has to contain the `Confirm` command and may contain additional commands (such as logging or switching via set command).

Database path: /EVENTS/EventHandler/System

```
<EventHandler>
   <System>
     <Confirmation>
         <Confirm _="&#xae;~/_ConfirmID"/>
         <Log _="Event">
              <ConfirmID _="&#xae;~/_ConfirmID"/>
         </Log>
     </Confirmation>
     ...
   </System>
   ...
<EventHandler>
```

The Confirm command performs the confirmation of the message identified by the confirmation ID

The event parameter _ConfirmID is generated from the fingerprint

2. The second way to invoke the confirmation event is to invoke it directly, using a `DoOn` command (chapter 2.4.6.3) or as an event in a command message (chapter 8) or via port change or service button (chapter 6).

Example

The service button has to be pressed by the service personnel to confirm the alarm and log the time of arrival.

Database path: /EVENTS/EventHandler/System

ConfirmID of EventHandler

```
<EventHandler>
    <System>
       <OnButton>
         <Confirm _="101"/>
         <Log _="Event">
              <Service _="Pressed upon alert by message 101"/>
         </Log>
       </OnButton>
     ...
   </System>
   ...
<EventHandler>
```

***Set** - Set System Property*	
Syntax	**`<Set _="`*`Path`*`" value="`*`Value`*`"/>`**
Description	Set the *Value* of the system properties referred by the *Path*. Note: There are many System Properties which are read only! The system properties are the set of data describing a FP Gateway. This includes administrative information like version numbers, licenses etc. which are defined at the creation time of the firmware, as well as information on the hardware configuration and the system state. The system state includes the system time, the system mode the states of the I/O ports, PLC variables etc. The configuration settings defined by the `SetConfig` are a part of the system state and therefore a part of the system properties. They can therefore also be accessed by the `Set` command. The difference to the `SetConfig` command is the way the data is addressed and the structure of the data set. Both commands use a slash separated path to address the data but "`Set`" addresses a single value only where "`SetConfig`" addresses complex values, for example a complete attribute group. A second difference is in the data itself. All System Properties have a unique address defined by their path. Configurations contain parts which have no unique addresses: For example the PLC "`External`" could contain several "`Devices`" on a "`Bus`" which all have the same tag name "`Device`". In this case an element can't be addressed uniquely by a path. Therefore, not all elements of the configuration can be addressed by the Set command. Use `SetConfig` instead.
Parameters	***Path:*** Path which addresses the system properties. See chapter 14 for details on system properties. ***Value:*** Value to set. The syntactical format depends on the value to set. See chapter 14 for details on system properties. The value may be created by references. The value string is limited to 80 characters.

Example	Set the relais output of a FP Gateway.
	
	`<Switch_0>`
	`   <Set _="/Process/MB/IO/Q/P2" value="1"/>`
	`<Switch_0>`

Log Command	

Syntax	**`<Log _="LogfileName">`**
	`   LogData`
	`</Log>`

Description	Creates an entry like this in the journal database:
	
	`<ID_nnn time=_"TimeStamp">`
	`   LogData`
	`</ID_nnn>`
	
	nnn: Unique ID to address the log entry.
	
	TimeStamp: System time when the log entry has been written.
	
	Note:
	The Log command can only be used for logfiles defined with the content type "XML".

Elements	*LogData:* XML formatted data to be logged. *LogfileName:* Name of the logfile to be used. Must be defined in the LogDefinition database. Note: You can insert attributes with references to any system property like `<Input_P4 _="&#xae;/Process/MB/IO/I/P4"/>` The reference will then be replaced by the corresponding value.

Example	Event handler logging the last power off and last power on time.
	
	`<SupportLog>`
	`   <Log _="SupportLog">`
	`      <PowerOff _="&#xae;/TIMES/PowerOffTime;"/>`
	`      <PowerOn _="&#xae;/TIMES/PowerOnTime;"/>`
	`   </Log>`
	`</SupportLog>`

BinLog Command	

Syntax	**`<BinLog _="LogfileName">`**
	`   <ValueName _="Value"/>`
	`   <ValueName _="Value"/>`
	`   ...`
	`</BinLog>`

Description	Creates a binary logfile entry with the structure of a given record. Note: The BinLog command can only be used for logfiles defined with the content type "binary" and an assigned record.

Elements	*LogfileName:* Name of the logfile to be used. Must be defined in the LogDefinition database. *ValueName:* Name of value defined in record database (optional) Value: Value to be written into the structure. (optional)

Example	Event handler logging a digital input if the FP Gateway.
	
	`<Datalogging_1_Log>`
	`   <BinLog _="Datalogging_0">`
	`      <Input4 _="&#xae;/Process/MB/IO/I/P0;"/>`
	`   </BinLog>`
	`</Datalogging_1_Log>`

Process Command	

Syntax	**`<Process>`**
	`   `*`Instruction List`*
	`</Process>`
	or
	`<Process _="`*`ProcessVariableName`*`"/>`

Description	Processes the instructions of the given instruction list or calculate the specified process variable.

Elements	*Instruction List:*
	List of instructions calculating the value of the process variable (for a description of instructions see Configuring Process Variables chapter 6.2). Strings are currently not supported.
	
	Note:
	To process a variable given by the event parameter, you can use the data path (~/) as address parameter of the instruction:
	
	Assume the event command:
	`<DoOn _="Switch_1">`
	`   <PortValue _="1"/>`
	`</DoOn>`
	
	The following event handler sets the port P1 to the value given by the event command.
	
	`<Switch_1>`
	`   <Process>`
	`      <LD _="&#xae;~/PortValue"/>`
	`      <ST _="/Process/MB/IO/Q/P1"/>`
	`   </Process>`
	`</Switch_1>`
	
	The reference *must not* be ended by a semicolon in this case! Alternative references are not supported in RPN instructions.
	
	ProcessVariableName:
	Name of a process variable configured in `/PROCCFG/ProcessVars/`. The process variable must include the "`Process`" command (instead of "Value") to be calculated on trigger (see chapter 6.3).

Example	Event handler sets the output port P4, logs this port and sends a message:
	
	`<Switch_0>`
	`   <Process>`
	`      <LD _="1"/>`
	`      <ST _="/Process/MB/IO/Q/P4"/>`
	`   </Process>`
	`   <Log _="Datalogging_1_Log">`
	`      <Action _="Port set"/>`
	`      <Portstate _="&#xae;/Process/MB/IO/Q/P4; "/>`
	`   </Log>`
	`   <SendMail _="MessageJobTemplates/Switch_0"/>`
	`</Switch_0>`
	
	Event handler triggers a process variable to increase a counter:
	
	`<Increase>`
	`   <Process _="Counter"/>`
	`</Increase>`
	
	The referenced process variable may look like this:
	
	`<CounterValue def="0"/>`
	`<Counter>`
	`   <Process>`
	`      <LD _="/Process/PV/CounterValue"/>`
	`      <ADD _="1"/>`
	`      <ST _="/Process/PV/CounterValue"/>`
	`   </Process>`
	`</Counter>`

Delay Command	

Syntax	***`Instruction`***
	`<Delay _="`*`Xs`*`"/>`
	`Instruction`

Description	Includes a delay between two instructions.
Elements	*Instruction:* EventHandler command, e.g. "SendMail" or "Set" etc. *Xs:* Time in seconds (1-60)

Example	Event handler sets the PLC variable 1, waits 5 seconds and processes thereafter the SendMail.
	
	`<Switch_1>`
	`   <Set _="/Process/Bus1/Device_0/Variable_1" value="1"/>`
	`   <Delay _="5s"/>`
	`   <SendMail _="MessageJobTemplates/Switch_1"/>`
	`</Switch_1>`
	
	This may be useful if a "Set" of a PLC variable is initiated by an incoming message and the "SendMail" should send a confirmation back to the sender including the value read after 5s to verify the event Without the delay the answer may include the old value because the PLC-protocol was not fast enough to process the command before creating the confirmation message.

Confirm Command	
Syntax	**`<Confirm _="ConfirmID"/>`**
Description	Used to confirm a message waiting for acknowledge (see SendMail parameter "OnTimeout").
Elements	*ConfirmID:* ID given in SendMail command or "*" to confirm all jobs.

Example	This EventHandler confirms a SendMail command waiting for acknowledge. The requested ConfirmID was 99.
	
	`<Confirmation_Event>`
	`    <Confirm _="99"/>`
	`</Confirmation_Event>`
	
	The Confirm command is mostly used by the system events "OnButton" and "Confirmation" (see chapter 3.7.3).

SetConfig Command	
Syntax	**`<SetConfig/>`**
Description	EventHandler command to change databases via incoming Email. The incoming Email has to be in "plain text" format. Disable any Rich-Text, HTML or quoted printable format option in your email program if you send a message to the FP Gateway. For further information, see chapter 9.4.

Example	An incoming message with subject "Password LoadDatabase" (Password: see chapter 9.5) and a database included in the message body will be processed by this EventHandler:
	`<Switch_0>`
	`    <SetConfig/>`
	`</Switch_0>`

POP3Query Command	
Syntax	**`<POP3Query/>`**
Description	Queries a configured POP3 account (see chapter 3.5) for new emails to process incoming messages (see chapter 9.3.4). There must be a delay of 20s between two POP3 queries. Shorter retries are not processed.

Example	This EventHandler may be triggered periodically via scheduler to query a POP3 account for new emails to process:
	
	`<Switch_1>`
	`    <POP3Query/>`
	`</Switch_1>`

Clear Command	
Syntax	**`<Clear Log="Logfiles"/>`**
Description	Deletes the content of one or several logfiles. (see chapter 2.4.6.6).
Elements	*Logfiles:* Logfile or list of logfiles to be deleted. To delete several Logfiles with one command, separate the logfile names by comma. Use an asterisk "*" to delete all logfiles.
	This EventHandler may be triggered on OnOK cascading after the logfile was sent via email:

Example	`<Datalogging_0_Clear>`
	`    <Clear Log="Datalogging_0"/>`
	`</Datalogging_0_Clear>`

Reset Command	
Syntax	***`<Reset/>`***
Description	Processes a "Reset keep" of the device (see chapter 2.4.6.1). Reset will be executed with a delay of 10s.
Elements	No elements

Example	This EventHandler may be triggered periodically via scheduler to reset the device.
	
	`<Switch_2>`
	`    <Reset/>`
	`</Switch_2>`

INetTime Command	
Syntax	***`<InetTime/>`***
Description	Queries an Internet TIME-Server to synchronize the RealTimeClock of the device.
Elements	No elements

Example	This EventHandler may be triggered once a month via scheduler to synchronize the FP Gateway clock with an Internet TIME server:
	
	`<Switch_4>`
	`    <INetTime/>`
	`</Switch_4>`

SetTime Command	
Syntax	**`<SetTime _="`*`Time`*`" TimeDiff="`*`Difference`*`"/>`**
Description	Sets the FP Gateway clock (RTC) to the given value. May be used to synchronize the FP Gateway time with the PLC time.
Elements	*Time:* Time string or reference to time string with following format: *YYYY/MM/DD,hh:mm:ss* *Difference:* Difference between the "Time" value and the time to set. The time zone provided in the /USER/USER database will be added to the difference. +/-HHMM

Example	This EventHandler copies the value of the PLC time variable into the FP Gateway RTC and adds one hour (/USER/USER/Timezone="+0000"):
	
	`<Switch_5>`
	`    <SetTime _="&#xae;/Process/Bus1/Dev0/Clock;"`
	`    TimeDiff="+0100"/>`
	`</Switch_5>`

S0_Sync Command	
Syntax	***`<S0_Sync/>`***
Description	This command is only used together with the S0-interface (see chapter 6.7). It generates a synchronization impulse by the device (e.g. via scheduler) instead of

	using an external synchronization impulse. The created synchronization impulse copies the counted S0 impulses into the counter variable.
Elements	No elements
Example	This EventHandler creates an synchronization impulse:

```
<Switch_6>
   <S0_Sync/>
</Switch_6>
```

Beep Command	
Syntax	**`<Beep _="melody" duration="length"/>`**
Description	This command activates the speaker.
Elements	***melody:*** selects the melody 1 – 6
	1 constant
	2 rhythmical same tone
	3 fast beeping
	4 fast siren
	5 slow siren
	6 rhythmical different tones
	length: selects the tone duration (in ms)
Example	This EventHandler switches the FP Gateway speaker on for 10 minutes:

```
<Switch_8>
   <Beep _="4" duration="600000"/>
</Switch_8>
```

This EventHandler switches the FP Gateway speaker off (if active):

```
<Switch_9>
   <Beep _="4" duration="0"/>
</Switch_9>
```

GetJobs	
Syntax	**`<GetJobs _="logfile"/>`**
Description	This command writes a list of job groups and currently active jobs into the specified logfile.
Elements	***logfile:*** Logfile in which the list will be written.
Example	This EventHandler writes the job list into the Event log:

```
<Switch_10>
   <GetJobs _="Event"/>
</Switch_10>
```

The logfile entry will have the same syntax as the GetJob command result.

DeleteJobs	
Syntax	**`<DeleteJobs _="logfile"/>`**
Description	This command deletes all waiting (not active) jobs and writes a list of them into the specified logfile.
Elements	***logfile:*** Logfile in which the list will be written.
	This EventHandler deletes all waiting jobs and writes the job list into the Event log:

Example

```
<Switch_11>
   <DeleteJobs _="Event"/>
</Switch_11>
```

The logfile entry will have the same syntax as the GetJob command result.

TransMode	
Syntax	**`<TransMode format="`*`SerialFormat`*`" local="`*`localSerialFormat`*`" baud="`*`Baudrate`*`" com="`*`comport`*`" handshake="`*`Handshake`*`" wait="`*`timeout`*`"/>`**
Description	Switches the FP Gateway to a local transparent mode. Data from COM1 will be routed to the selected extension com port (COM2/COM3). Note: This kind of transparent mode will not be aborted by a Plug&Play sequence and there is also no idle timeout. Don't forget to create a TransModeClose event, otherwise the device has to be manually switched off/on to leave the transparent mode.
Elements	See chapter 2.4.6.1 for valid command parameters (exept `keep`).

Example	This System EventHandler switches the FP Gateway into transparent mode if service button is pressed and will leave it if pressed again:
	
	`<System>`
	`   <OnButton>`
	`      <TransMode baud="2400" local="8E1" format="8E1`
	`     com="COM3"/>`
	`      <If _="/Process/MB/TransMode">`
	`     <TransModeClose/>`
	`      </If>`
	`   </OnButton>`
	`</System>`

TransModeClose	
Syntax	**`<TransModeClose/>`**
Description	Tells the FP Gateway to leave the transparent mode.

Example	This System EventHandler switches the FP Gateway into transparent mode if service button is pressed and will leave it if pressed again:
	
	`<System>`
	`   <OnButton>`
	`      <TransMode baud="2400" local="8E1" format="8E1"`
	`com="COM3"/>`
	`      <If _="/Process/MB/TransMode">`
	`     <TransModeClose/>`
	`      </If>`
	`   </OnButton>`
	`</System>`

Connect	
Syntax	**`<Connect/>`**
Description	Establishes an ISP connection. The connection will be kept online depending on the KeepConnected ISP database setting (see chapter 3.5).

Example	This System EventHandler establishes an internet connection as soon as the GSM device is registered to the GPRS network.
	
	`<System>`
	`   <GPRSPrepared>`
	`      <Connect/>`
	`   </GPRSPrepared >`
	`</System>`

GPRSDisconnect	
Syntax	**`<GPRSDisconnect delay=”Delay” />`**
Description	Disconnects a GPRS / UMTS / LTE connection. The delay is optional.

Example	This System EventHandler closes an open internet connection immediately.
	
	`<System>`
	`   <MobileClose>`
	`      <GPRSDisconnect/>`
	`   </MobileClose>`
	`</System>`

WriteFile	
Syntax	**`<WriteFile _="`*`Template`*`" File="`*`Filename`*`" FileExistsOperation="`*`Mode`*`"/>`**
Description	Writes the content generated by the text template to the SD card.
Elements	*Template:* Name of the message job template which is used to generate the job for this event. See chapter 3.9. *Filename:* Name of the file to be created/appended on the SD-Card. Must be in 8.3 notation. *Mode:* Defines what to do if the filename already exists. keep: The file will not be overwritten, writing aborted. (default) override: Overwrite file. append: Append data to the end of the existing file.

Example	This EventHandler copies the content of a log file to the SD-Card. The unique filename is created by the time in HEX format.
	
	`<Switch_12>`
	`   <WriteFile _="MessageJobTemplates/Switch_12"`
	`    File="&#xae;/TIMES/HEXDATE.CSV"/>`
	`</Switch_12>`

CheckJobConditions	
Syntax	**`<CheckJobConditions/>`**
Description	Checks the condition (variable) of a delayed SendMail job. If the condition is TRUE, the message will be sent. If the condition is FALSE, delay will be continued.
Example	This EventHandler checks the condition of delayed SendMail jobs: `<Switch_13>` `<CheckJobConditions/>` `</Switch_13>`

Start GPRS-connection	
Syntax	**`<StartGPRS>`** **`<Connect/>`** **`</StartGPRS>`**
Description	The GPRS / UMTS / LTE connection can be started by calling an Event that needs to establish an internet connection (sending an email or with an EventHandler executing the command <Connect/>).

Stop GPRS-connection	
Syntax	**`<StopGPRS>`** **`<GPRSDisconnect delay="delaytime"/>`** **`</StopGPRS>`**
Description	The GPRS / UMTS / LTE connection can be stopped, by sending the TiXML command [<GPRSDisconnect delay="delaytime" ver="v "/>] or with an EventHandler executing the command <GPRSDisconnect/>.
Elements	delaytime Delays the execution of the command. The time has to be set in ms (without scale).

3.7.2 Conditions

The "IF" instruction is used to prevent the execution of events in special conditions. For example it may be used to deactivate scheduled data logging during transparent mode or failure of the connected external device.

If instruction	
Syntax	**`<If _="Condition">`** ***`EventHandler commands`*** **`</If>`**
Description	Processes the enclosed EventHandler commands only if the condition equals "1".
Elements	*Condition* Path to a bit variable, e.g. ProcessVar *EventHandler commands:* List of EventHandler commands (for a complete list see chapter 3.7.1).
Example	Data logging is only processed if PLC communication is active. `<Datalogging_0_Log>` `<If _="/Process/Bus1/Device_0/DeviceState">` `<Log _="Port" >` `<PortLog1 _="&#xae;/Process/Bus1/Device_0/Word01"/>`

```
            <PortLog2 _="&#xae;/Process/Bus1/Device_0/
               Word02"/>
         </Log>
      </If>
</Datalogging_0_Log>
```

IfNot instruction	

Syntax	**`<IfNot _="Condition">`**
	   `EventHandler commands`
	`</IfNot>`

Description	Processes the enclosured EventHandler commands only if the condition is not 1.
Elements	*Condition* Path to a bit variable, e.g. ProcessVar *EventHandler commands:* List of EventHandler commands (for a complete list see chapter 3.7.1).

Example	Data logging is only processed if ethernet connection is lost.
	
	`<Datalogging_0_Log>`
	`   <IfNot _="/Ethernet/LinkState">`
	`      <Log _="Port" >`
	`         <PortLog1 _="&#xae;/Process/Bus1/Device_0/`
	`           Word01"/>`
	`         <PortLog2 _="&#xae;/Process/Bus1/Device_0/`
	`           Word02"/>`
	`      </Log>`
	`   </IfNot>`
	`</Datalogging_0_Log>`

3.7.3 System events

Inside EventHandler database a group "System" exists, that holds EventHandler that are not triggered by "DoOn" command or "EventState" but by special system conditions.

System EventHandler	

Syntax	**`<System>`**
	   `<SystemEventName>`
	      `EventHandler commands`
	   `<SystemEventName>`
	`</System>`

Description	Processes the enclosed EventHandler commands triggered by special system conditions.
Elements	*SystemEventName* Type of system event: GPRSPrepared Will be triggered as soon as the GSM device is registered to the GPRS network. Requires `GPRS=On` in USER database (see chapter 3.3 and 3.5). OnButton Will be triggered as soon as the service button is pressed. Confirmation Will be triggered as soon as the device receives a message with valid fingerprint (see chapter 3.7.1) POPInvalidPassword

	TixiInvalidPassword SMSInvalidPassword CGIInvalidPassword Events triggered during processing of received messages or CGI calls, if password was invalid (see chapter 9.3.1). POPInvalidEvent TixiInvalidEvent SMSInvalidEvent CGIInvalidEvent Events triggered during processing of received messages or CGI calls, if command was invalid or if failure during processing a valid command (see chapter 9.3.1). *EventHandler commands:* List of EventHandler commands (for a complete list see chapter 3.7.1).

Example	This System EventHandler establishes an internet connection as soon as the GSM device is registered to the GPRS network.
	
	`<System>`
	`   <GPRSPrepared>`
	`      <Connect/>`
	`   </GPRSPrepared>`
	`</System>`

3.8 Message Text Template

This template can be used for all message types but typically SMS and pager templates will only contain a subject line. The templates contains instructions for the Job Generator which creates a mesaage from the template.

Database path: /TEMPLATE/UserTemplates

Message Text Template	

Syntax	***`<TemplateName>`***
	`   Instruction List`
	`</TemplateName>`

Description	Template, creating a message text. The Job Generator processes the instructions of the template from top to bottom. The result is textual output into the message body and subject.
Elements	*TemplateName :* Name of the template. *Instruction List:* List of instructions to be processed by the template processor. See the following paragraphs for information on these instructions.

Example	Typical template with subject and body. References will be resolved or lines skipped if they include unresolveable references.A signature will be copied to the end of the message.
	
	`<UserTemplates>`
	`   <Message_0>`
	`      <Subject>`

```
                        <C _="This is the subject line"/>
                     </Subject>
                     <Body>
                        <E _="Beginning of message body"/>
                        <E _="enter some more lines"/>
                        <L _=""/>
                        <E _="Date: &#xae;/TIMES/Date,?; "/>
                        <E _="----------------------------------------
                        "/>
                        <Include
                        _="/D/UserTemplates/LocationText/Email"/>
                     </Body>
                  </Message_0>
               </UserTemplates>
```

Note:
For SMS and pager messages, a single line or several lines without line break are used. The line is defined by the message job template as subject (see Message Job Templates chapter 3.9). If you want to include logfiles into the message text, see chapter 4.3.

The template processor uses the following instructions:

***Write a text line** – abort on error*	
Syntax	**`<L _="RefText"/>`**
Description	Writes a text string with a CR/LF at the end of the line. The text can contain references to other attributes. These references are replaced by the values of the attributes. If the FP Gateway can't resolve the attribute, it will stop processing the event.
Elements	***`RefText:`*** Text to write. The text could include some references to system properties or parameters which are placed into the client event messages. In the output line, these references are replaced by the values of the referred attributes. A reference starts with the '**&#xae;**' string and ends with a semi colon or the end of the tag.

Example	Writes the line 'Temperature of Barn: 10 °C'. The value '10' is inserted from the client message data attribute temperature. The degree "°" is written as an entity.
	
	Client event message:
	`[<DoOn _="Alarm_0">`
	`   <Temperature _="10"/>`
	`</DoOn>]`
	
	Template processor instruction:
	`<L _="Temperature of Barn: &#xae;~/Temperature;&#xb0;C"/>`
	
	Lines in the message:
	`Temperature of Barn: 10°C`

***Write a text line** – continue on error*	
Syntax	**`<E _="RefText"/>`**
Description	Writes a text string with a CR/LF at the end of the line. The text can contain references to other attributes. These references are replaced by the values of the

attributes. If the FP Gateway can’t resolve the attribute, it will skip the complete line and continue processing the next line.

Elements

RefText:
Text to write. The text could include some references to system properties or parameters which are placed into the client event messages. In the output line, these references are replaced by the values of the referred attributes. A reference starts with the '**&#xae;**' string and ends with a semi colon or the end of the tag.

Example

Writes the line 'Temperature of Barn: 10 °C'. The value '10' is inserted from the client message data attribute temperature. The degree "°" is written as an entity. Due to a missing attribute, the FP Gateway can’t resolve all parameters.

Client event message:
```
[<DoOn _="Alarm_0">
   <Temperature _="10"/>
</DoOn>]
```

Template processor instruction:
```
<E _="Temperature of Barn: &#xae;~/Temperature;&#xb0;C"/>
<E _="Barn: &#xae;~/Barn;  "/>
```

Lines in the message:
```
Temperature of Barn: 10°C
```

Line two will be skipped, because attribute "Barn" doesn’t exist.

***Write a text line** – no CR/LF, abort on error*

Syntax

`<S _="RefText"/>`

Description

Writes a text string without a CR/LF at the end of the line. The text can contain references to other attributes. These references are replaced by the values of the attributes. If the FP Gateway can’t resolve the attribute, it will stop processing the event.

Elements

RefText:
Text to write. The text could include some references to system properties or parameters which are placed into the client event messages. In the output line these references are replaced by the values of the referred attributes. A reference starts with the '**&#xae;**' string and ends with a semi colon or the end of the tag.

Example

Writes the line 'Temperature of Barn: 10°C Barn 12'. The values are inserted from the client message data attribute barn and temperature. The degree "°" is written as an entity.

Client event message:
```
[<DoOn _="Alarm_0">
   <Barn _="12"/>
   <Temperature _="10"/>
</DoOn>]
```

Template processor instruction:
```
<S _="Temperature of Barn: &#xae;~/Temperature;
  &#xb0;C "/>
<S _="Barn: &#xae;~/Barn;  "/>
```

Line in the message:
```
Temperature of Barn: 10°C Barn 12
```

***Write a text line** – no CRLF, continue on error*	
Syntax	**`<C _="RefText"/>`**
Description	Writes a text string without a CR/LF at the end of the line. The text can contain references to other attributes. These references are replaced by the values of the attributes. If the FP Gateway can't resolve the attribute, it will skip the complete line and continue processing the next line.
Elements	***`RefText:`*** Text to write. The text could include some references to system properties or parameters which are placed into the client event messages. In the output line these references are replaced by the values of the referred attributes. A reference starts with the '**`&#xae;`**' string and ends with a semi colon or the end of the tag.

Example	Writes the line 'Temperature of Barn: 10°C'. The value '10' is inserted from the client message data attribute temperature. The degree "°" is written as an entity. Due to a missing attribute, the FP Gateway can't resolve all parameters.
	
	Client event message:
	`[<DoOn _="Alarm_0">`
	`   <Temperature _="10"/>`
	`</DoOn>]`
	
	Template processor instruction:
	`<C _="Temperature of Barn: &#xae;~/Temperature;`
	`  &#xb0;C "/>`
	`<C _="Barn: &#xae;~/Barn; "/>`
	
	Line in the message:
	`Temperature of Barn: 10°C`
	
	Line two will be skipped, because attribute "Barn" doesn't exist.

***Include** – Include another text template*	
Syntax	**`<Include _="Path to Text Template"/>`**
Description	Includes another text templates into the message. May be used to add a signature to each message.
Elements	***`Path to Text Template:`*** XML-Path to the template to be included.

Example	Includes the template "`LocationText`" into the message text
	Signature template:
	`<LocationText>`
	`   <L _="Location: "/>`
	`   <L _="InovoLabs GmbH"/>`
	`   <L _="Berlin"/>`
	`</LocationText >`
	
	Text template with reference to `LocationText`:
	`<Message_0>`
	`   <L _="Enter your message text here"/>`
	`   <Include _="/D/UserTemplates/LocationText"/>`
	`</Message_0>`

***IncludeSP** – Include system properties*

Syntax

```
<IncludeSP _="Path to system property" AddInfo="Info"
ViewProperties="Properties" format="Format"/>
```

Description

Includes the system properties tree or parts of it into the message.

Elements

Path to system property:
XML-Path to the system properties to be included (see chapter 14).

empty:	complete system properties tree
/PATH:	system properties branch or single value

Info:

Error:	Displays the error state of the variable(s) instead of its value(s).

Properties:

Error:	Displays ErrorClass and ErrorValue additionally to the value(s)
Exp:	Displays the exponent of the value(s)

Format:
Formats the output value (see chapter 6.5.2 for instructions)

Example

Includes the complete system property tree (similar to `[<Get/>]` command) into the message text:

```
<Message_0>
   <IncludeSP _=""/>
</Message_0>
```

Includes the process tree with Error codes:

```
<Message_1>
   <IncludeSP _="/Process/" AddInfo="Error"/>
</Message_1>
```

Includes the process tree with additional error and exponent information:

```
<Message_2>
  <IncludeSP _="/Process/" ViewProperties="Error,Exp"/>
</Message_2>
```

Includes and reformats the first digital input:

```
<Message_3>
 <IncludeSP _="/Process/MB/IO/I/P0" format="?on,off" />
</Message_3>
```

***IncludeLog** – Include a XML log file*

Syntax

```
<IncludeLog _="LogFileName" range="entryrange"/>
```

Description

See chapter 4.6 for complete reference and examples.

***IncludeLogTXT** – Include an reformat a log file*

Syntax

```
<IncludeLogTXT _="LogFileName" range="entryrange"
fillInterval="interval" maxInterval="tolerance"
fillText="string" Viewset="variables" Formats/>
```

Description

See chapter 4.6 for complete reference and examples.

CopyDatabase – Include XML databases	
Syntax	**`<CopyDatabase _="`*`Path to database`*`" flags="`*`Dereferer`*`" dest="`*`Destination`*`"/>`**
Description	Copies the defined XML database(s) into the message text.
Elements	*Path to database:* Path to the XML database to be copied into the message text / copies the complete TiXML project /DATABASE/Group copies the specified database or groups *Dereferer (optional):* Defines if references are to be processed empty: References are not processed (default) Deref: References are processed *Destination (optional):* Defines XML tags enclosing the copied database.

Example	Copies the EventHandler database into the message text:
	`<Message_1>`
	`   <L _="EventHandler Database:"/>`
	`   <L _=""/>`
	`   <CopyDatabase _="/EVENTS/D/EventHandler"/>`
	`</Message_1>`
	
	Copies the complete TiXML project into the message text, enclosed by XML-Tags "CONFIG":
	`<Message_2>`
	`   <CopyDatabase _="/" dest="CONFIG"/>`
	`</Message_2>`

CheckSum – Create a checksum	

Syntax	**`<StartCheckSum _="CRC32"/>`**
	`   `*`Instruction List`*
	`<StopCheckSum _="`*`Format`*`"/>`

Description	Creates a CRC32 checksum over the text created by the enclosured instructions (including CRLFs). The checksum can be included into the message text using a reference to the system variable `"_CheckSum".` CRC32 calculation: Width: 32Bit Polynomal: 04C11DB7 Init Value: FFFFFFFF Reflection: In/Out deactivated XOR Out: 00000000
Elements	*Format:* Checksum format X: Hexa decimal D: Decimal
	Creates a CRC32 checksum over the message text "Hello World!": `<Message_2>`

Example

```
       <StartCheckSum _="CRC32"/>
       <E _="Hello World!"/>
       <StopCheckSum _="X" />
       <E _="CRC32=&#xae;~/_CheckSum;"/>
    <Message_2>
```

Checksum: `B43B5AC1`

3.9 Message Job Template

A typical reaction to a client event is the sending of one or more messages. Each of these is created by a message job template (MJT) which combines the sender, recipient and message text when the event occurs. This is done by creating a message job which is added to the message queue of a Job Generator - similar to a computers printer queue.

These templates are defined in the 'TEMPLATE' database. Each template is an attribute group which has a unique name which has to be written as parameter of the 'SendMail' command in the event handler configuration. The owned attribute (_="") defines the message transport type like email, SMS, Fax etc..

We recommend using not more than 100 message job templates, as otherwise the processing speed may drop below reasonable levels.

Templates to create message jobs for a certain client event:

Database path: /TEMPLATE/MessageJobTemplates

```
<MessageJobTemplates>
   <Alarm_0 _="SMTP">
      <Recipient _="/D/AddressBook/Contact_0"/>
      <Sender _="/D/AddressBook/MySelf"/>
      <Body _="/UserTemplates/Message_0/Body"/>
      <Subject _="Barn is out of temperature"/>
   </Alarm_0>

   <Alarm_1 _="SMS">
      <Recipient _="/D/AddressBook/Contact_1"/>
      <Sender _="/D/AddressBook/MySelf"/>
      <Subject _="Barn &#xae;~/Barn; is out of temperature.
   Temp=&#xae;~/Temperature; C. "/>
   </Alarm_1>

   <Alarm_2 _="GSMSMS">
      <Recipient _="/D/AddressBook/Contact_0"/>
      <Sender _="/D/AddressBook/MySelf"/>
      <Subject _="" Path="/UserTemplates/Message_1/Subject"/>
   </Alarm_2>
</MessageJobTemplates>
```

Message Job Templates

Message Job Template	
Syntax	***<TemplateName _="TransportTypeTemplate">*** ***List of Variables*** ***</TemplateName>***
Description	Attribute group which defines the creation of a message job for a certain event.

Elements	*TemplateName:* Name of the template. This is the parameter of the 'SendMail' command in the event handler configuration and must be unique within the MessageJobTemplate group. *List of Variables:* List of attributes defining some variables depending on the predefined templates. *TransportTypeTemplate*: SMTP creating a SMTP message job. SMS creating a SMS message job via landline modem. TextFax creating a FAX message job. GSMSMS creating a SMS message job via a GSM modem. CityRuf creating a pager message. URLSend creating a HTTP notification (GET) CBIS creating a CBIS notification WriteFile creating a job to write to the SD-Card
Example	This message job template creates a SMTP message job: (see below)

This message job template creates a SMTP message job:

Variables

```
<Alarm_0 _="SMTP">
   <Recipient _="/D/AddressBook/Contact_0"/>
   <Sender _="/D/AddressBook/MySelf"/>
   <Body _="/UserTemplates/Message_0/Body"/>
   <Subject _="Barn is out of temperature"/>
</Alarm_0>
```

Note:
An SMTP message can be sent to more than one recipient. This can be achieved by more than one 'Email' entry in the referred AddressBook contact instead of adding several "Recipients" to the MJT

The subject line of the message may be written directly into the message job template attribute "Subject", but for software compatibility reasons we recommend to use the "path" attribute to an user template instead (see next note):

```
<Alarm_0 _="SMTP">
   <Recipient _="/D/AddressBook/Contact_0"/>
   <Sender _="/D/AddressBook/MySelf"/>
   <Body _="/UserTemplates/Message_0/Body"/>
   <Subject _="" Path="/UserTemplates/Message_0/Subject"/>
</Alarm_0>
```

The variables necessary for a message job template are depending on the message type. Typically you have to add the sender and receiver address, the template for the body and the subject but some message types don't need a body or even a text at all.

Note:
The subject can be defined by three different methods:

<u>Direct (only valid for this MJT, limited to 385 characters):</u>
Subject is written directly into the MJT:
`<Subject _="Barn is out of temperature"/>`

<u>Reference (useable for different MJTs, limited to 385 characters):</u>

	Subject is written into UserTemplates and referenced within MJT by `&#xae;`:
	`<Subject _="&#xae;/D/UserTemplates/Message_0/Subject;"/>`
	
	Referrenced UserTemplate:
	`<Message_0>`
	`<Subject _="Test"/>`
	`</Message_0>`
	
	<u>Path (useable for different MJTs, no character limitation):</u>
	Subject is written into UserTemplate and included via "Path":
	`<Subject _="" path="/D/UserTemplates/Message_0/Subject"/>`
	
	Referenced UserTemplate:
	`<Message_1>`
	`<Subject>`
	`<S _="385 characters text"/>`
	`<S _="add 385 characters text"/>`
	`<S _="add 385 characters text"/>`
	`...`
	`</Subject>`
	`</Message_1>`
	
	If there is additional text within _="" it will be added in front of the UserTemplates text.

SMTP

Name	*Description*
Recipient	Path to the address book entry including the email address for the receiver of the SMTP message (inserted in the 'To' field of the message).
Sender	Path to the address book entry including the email address for the sender (inserted in the 'From' field of the message).
Subject	The subject text of the message (inserted in the 'Subject' field of the message).
Body	The Body contains the main text of the message. This is what the receiver of the mail will read and it should contain all the necessary information to react to the event. The Body field contains a path to the message text template where the message text is defined. Therefore you can use one of these message text templates for several message job templates.
Attachments	Optional attachment with logged data. See chapter 4.6.2.

TextFax

Name	*Description*
Recipient	Path to the address book entry including the fax number for the receiver of the fax message
Sender	Path to the address book entry including fax number for the sender (inserted in the header of the fax).
Subject	see SMTP
Body	see SMTP

SMS/GSMSMS

Name	*Description*
Recipient	Path to the address book entry including the SMS number of the receiver of the SMS message.
Sender	Path to the address book entry including the phone number of the sender of the SMS message.
Subject	Defines the SMS message text (max. 160 characters).
Body	not used

CityRuf

Name	*Description*
Recipient	Path to the address book entry including the pager number of the receiver of the CityRuf pager call.
Sender	not used
Subject	Defines the pager message text.
Body	not used

CBIS

Name	*Description*
Recipient	Path to the address book entry including the email address for the receiver of the SMTP message (inserted in the 'To' field of the message).
Sender	Path to the address book entry including the email address for the sender (inserted in the 'From' field of the message).
Subject	not used, subject with IP-adress link is automatically created
Body	not used, body will contain content of /USER/SITE_TAG database

URLSend

Name	*Description*
Recipient	Path to the address book entry including the URL to request.
Sender	not used
Subject	not used
Body	not used

WriteFile

Name	*Description*
Recipient	not used
Sender	not used
Subject	not used
Body	The Body field contains a path to the text template where the text is defined which will be written to the file on the SD card.

3.10 SMS Provider

The ISP database contains a section SMS_Provider which configures the access to SMS service centers. FP Gateways are supporting script gateways using TAP or UCP and gateways according to ETSI ES 201 912 (1TR140).

Database path: /ISP/SMS_Providers

SMS Provider	

Syntax	**<*ProviderName*>**
	    <Dialin _="*SMSCNumber*"/>
	    <Type _="*ProtocolType*"/>
	    <Script _="*ModemScript*"/>
	    <NumberFormat _="*Format*"/>
	    <SMS_ISDN _="*ISDNProtocol*"/>
	    <Pager_ISDN _="*ISDNProtocol*"/>
	    <SMS_Media _="*Media*"/>
	</*ProviderName*>

Description	Attribute group which specifies a gateway to send SMS.
Elements	*ProviderName* Name of the provider. This is the parameter of the 'SMS_Provider' or 'Pager_Provider' attrbute in the address book and must be unique within the SMS_provider group. *SMSCNumber* Phone number of the SMS center. *ProtocolType* This can be either SMS ETSI ES 201 912 (1TR140), only for PSTN modems Script TAP, UCP or GSM *ModemScript* The protocol for the SMS transmission (only if Type=Script). Can be either D1_TAP TAP with format 8N1 Mobilkom_A_TAP TAP with format 7E1 D2_UCP UCP GSM using the SMSC of the SIM card. *Format* Used to convert an canonical addressbook number into the format expected by the gateway, e.g. addressbook: +49-172-1234567 will be sent as: "national": 01721234567 "canonical": +491721234567 *ISDNProtocol* ISDN B-channel protocol used by SMS gateway. See chapter 3.5 ModemProtocol for supported values. *Pager_ISDN* ISDN B-channel protocol used by Pager gateway. See chapter 3.5 ModemProtocol for supported values. *Media* Must be ‘SMS’ for GSM providers.

Example

Modem Gateways (UCP, TAP):
The following example configures the access to the SMS modem gateway of the Vodafone Germany via ISDN:

```
<SMS_Provider_0>
  <Dialin _="+49-172-2278025"/>
  <Type _="Script"/>
  <Script _="D2_UCP"/>
  <NumberFormat _="national"/>
  <SMS_ISDN _="X.75-T.70"/>
</SMS_Provider_0>
```

PSTN gateways (ETSI ES 201 912, 1TR140):
This example configures the access to the SMS PSTN gateway of the german provider "AnnyWay":

```
<SMS_Provider_1>
  <Dialin _="+49-900-3266900"/>
  <Type _="SMS"/>
  <NumberFormat _="canonical"/>
</SMS_Provider_1>
```

GSM-SMSCs:
This example configures the access to the SMSC stored on the SIM card of your GSM device.

```
<GSM>
  <Dialin _="0"/>
  <Type _="Script"/>
  <NumberFormat _="canonical"/>
  <Script _="GSM"/>
  <SMS_Media _="SMS"/>
</GSM>
```

Some german SMS providers are preconfigured in our TiXML examples and TILA software.
Contact your local telephone company to get access to their SMS gateways (GSM, TAP, UCP, 1TR140).

3.11 Service center for incoming SMS

The ISP database contains a section "IncomingSMSCenter" which contains the callerIDs of the service center for incoming PSTN SMS. Three entries are possible (SMSC1-3).

Example

The following example configures the callerIDs for the german Telekom and the AnnyWay service center.

Database path: /ISP/IncomingSMSCenter

```
  <SMSC1 _="0193010" />
  <SMSC2 _="09003266900" />
```

3.12 Automatic Transmode

The FP Gateway is able to redirect some (callerID) or all incoming calls to one of its serial interfaces.

This offers transparent mode capability without need to send Login or TransMode commands, therefore it may be possible to use dial routines of the software that needs access to the device.

Database path: /ISP/AutoTransMode

Automatic TransMode	

Syntax	`<AutoTransMode>`
	    `<No`***`X`*** `_="`***`CID`***`" transmode="`***`comport`***`" format="`***`SerialFormat`***`" baud="`***`Baud Rate`***`" handshake="`***`Handshake`***`" wait="`***`timeout`***`"/>`
	`</AutoTransMode>`

Description	1. Switches the remote FP Gateway into transparent mode to route data to the selected com port. 2. It transforms the baud rate and the serial data format from the phone line to the values required by the connected client device. Note: This transparent mode of the remote FP Gateway becomes ended when the dialup connection drops. After this, the remote FP Gateway goes back in the TiXMLMode. Look into the manual of the dialing modem on how to disconnect connections (escape sequence). A transparent mode to the host port COM1 is blocked if a local login session is open.
Elements	*X:* Entry number. Possible values: 1 or 2 *CID:* CallerID used to trigger the automatic transmode ***comport:*** Specifies the COM port on the remote FP Gateway used for the connection. COM1 PLC port (labeled ***COM1 RS232***) (**default**) COM2 PLC port (labeled ***COM2 RS232*** or ***COM2 R-485/422***) (if available) COM3 M-Bus port (labeled ***M-Bus***) (if available) COM4 PLC port (***COM4 R-485/422***) (if available) *SerialFormat:* String which encodes the serial format that is used between modem and client device. It has the following syntax *(default "8N1"):* *DataBitsParityBitsStopBits* *DataBits* 8...8 data bits are used. 7...7 data bits are used. *ParityBits* N...No parity bit. E...Even parity. O...Odd parity. *StopBits* 1...one stop bit. 2...two stop bits. *Baud Rate:*

Used baudrate in bits per second (bps) **(default 9600).**

Handshake:
Used communication handshake.

None	communication without handshake
XONXOFF	software handshake
XONXOFFPASS	software handshake, XONXOFF forwarded to application
RTSCTS	hardware handshake with RTS CTS
DTRDSR	hardware handshake with DTR DSR
HALF	HalfduplexRS 485 communication
FULL	Fullduplex RS 485/422
HALFX	Halfduplex RS 485 communication with XON XOFF
FULLX	Fullduplex RS 485/422 with XON XOFF
noDTR	special DTR mode for Moeller and Mitsubishi PLCs

Note:
RS 485/422 communication is only valid on RS 485/422 interfaces.

timeout:
Specifies the time the FP Gateway will try to disable a PLC bus protocol on the remote com port (default: 20s).

Example

Establishes transparent mode to COM2 with 38400bps, data format 8O1 and hardware handshake if call with callerID 0301234567 is detected and (No2) establishes transparent mode to COM1 with 9600bps, data format 8E1 if call with callerID 0307654321 is detected:

```
<AutoTransMode>
   <No1 _="0301234567" transmode="COM2" format="8O1"
   baud="38400" handshake="RTSCTS" wait="60s"/>
   <No2 _="0307654321" transmode="COM1" format="8E1"
   baud="9600" handshake="none" wait="60s"/>
</AutoTransMode>
```

Establish transparent mode to COM1 with 9600bps, data format 8N1 on any call:

```
<AutoTransMode>
   <No1 _="*" transmode="COM1" format="8N1" baud="9600"
   handshake="none" wait="60s"/>
</AutoTransMode>
```

3.13 Internet Time synchronization

The FP Gateway uses a battery buffered Real Time Clock. Add the following configuration to synchronize the clock with an Internet Time Server via DayTime Protocol (TCP port 13):

Database path: /ISP/ISP/TimeServer

Internet Time synchronization with DayTime server	

Syntax	**`<TimeServer>`**
	   `<ServerName _="`*`address`*`"/>`
	   `<Protocol _="`*`protocol`*`"/>`
	   `<TimeDiff _="`*`difference`*`"/>`
	   `<TimeFormat _="`*`string`*`"/>`
	`</TimeServer>`

Description	Defines the time server to query, the used protocol, the time difference and the time format.
Elements	*address* Address of the internet time server. *protocol* Protocol used to query the time server. DAYTIME: DayTime protocol on TCP port 13 *difference* Time difference to GMT in coherence to the USER database TimeZone. Format: +HHMM (Default: +0000) Example Germany: The TimeZone setting in user database has to be +0100 GMT. To synchronize the time with a local time server, the Time Zone has to be subtracted from the server time, which means TimeDiff: -0100 *string* Format string that tells the device how the DAYTIME server response will look like. Following elements are available: y year m month d day h hour n minute s sec G month as string german E month as string english i ignore e.g.: Server response: `"11 MAY 2004 12:09:15 METDST"` TimeFormat: `"d E y h:n:s "`

Example	Time server settings to get GMT:
	`<TimeServer>`
	   `<ServerName _="time.nist.gov"/>`
	   `<Protocol _="DAYTIME"/>`
	   `<TimeDiff _="+0000"/>`
	   `<TimeFormat _="i y-m-d h:n:s i"/>`
	`</TimeServer>`

Scheduler for the DayTime synchronization (e.g. execute the DayTimeSync event every Monday):

```
<SyncTime _="SyncTime">
   <Weekday _="Mo"/>
</SyncTime>
```

You can also use this feature to change the clock to winter summer time (requires a local DAYTIME server with daylight saving time). Necessary scheduler:

```
<Summertime _="SyncTime">
   <Month _="3"/>
   <Day _="25-31"/>
   <Weekday _="Su"/>
   <Time _="02:00"/>
</Summertime>
<Wintertime _="SyncTime">
   <Month _="10"/>
   <Day _="25-31"/>
   <Weekday _="Su"/>
   <Time _="03:00"/>
</Wintertime>
```

Internet Time synchronization with NTP	

Syntax	`<TimeServer>`
	`   <ServerName _="servername"/>`
	`   <Protocol _="protocolname"/>`
	`   <TimeDiff _="timezone"/>`
	`</TimeServer>`

Elements	***servername*** name of the server ***protocolname*** server protocol ***timezone*** time zone

Example	`<TimeServer>`
	`   <ServerName _="ntps1-1.cs.tu-berlin.de"/>`
	`   <Protocol _="NTP"/>`
	`   <TimeDiff _="0000"/>`
	`</TimeServer>`

Scheduler for the NTP synchronization (e.g. execute the NTPSync event every day):

```
<NTPSync _="NTPSync">
   <Condition _="Condition/Pattern1day"/>
</NTPSync>
```

To start the time synchronization, a simple EventHandler will do the job:

```
<SyncTime>
   <INetTime/>
</SyncTime>
```

3.14 Ethernet

The FP IoT Gateway has a LAN interface for HTTP and TiXML access or email sending.
This chapter describes the TCP/IP configuration

Database path: /ISP/Ethernet

Ethernet configuration

Syntax

```
<Ethernet _="keep">
   <IP _="IP-address"/>
   <Mask _="Subnetmask"/>
   <Gateway _="GW-address"/>
   <FirstDNSAddr _="DNS-address"/>
   <SecondDNSAddr _="DNS-address"/>
   <HostName _="Host"/>
   <IP_2 _="IP-address"/>
   <Mask_2 _="Subnetmask"/>
</Ethernet>
```

Description

Defines the TCP/IP settings of the FP IoT Gateway. The second IP address can be used as target for the access to the device via VPN tunnel e.g.

Elements

keep

Flag to activate none volatile IP configuration written to EEProm.

persistent:	keep configuration in EEProm
empty:	don't keep confguration

IP-address

Static IP address of the FP IoT Gateway in "dotted quad format" or string "DHCP" to activate dynamic host configuration protocol.

Subnetmask

Subnet mask according to the IP address of the FP IoT Gateway. Can be omitted if DHCP is activated.

GW-address

Gateway IP address of the next router. Can be omitted if offered by DHCP server.

DNS-address

IP address of the DNS server. Can be omitted if offered by DHCP server.

Host

DHCP option 12 used for DDNS registration.

Example

Persistent IP configuration for private CLASS-C /24 network with a WAN router at 192.168.0.1.

```
<Ethernet _="persistent">
   <IP _="192.168.0.20"/>
   <Mask _="255.255.255.0"/>
   <Gateway _="192.168.0.1"/>
   <FirstDNSAddr _="192.168.0.2"/>
</Ethernet>
```

A second LAN IP address in the process path.

```
<Ethernet>
   <Link _="100"/>
   <LinkState _="1"/>
```

```
      <AssignedIP _="193.101.167.181"/>
      <SubnetMask _="255.255.255.128"/>
      <MAC _="00:11:E8:25:30:14"/>
      <Gateway _="193.101.167.193"/>
      <DNS_1 _="193.101.167.40"/>
      <AssignedIP2 _="10.251.198.158"/>
      <SubnetMask2 _="255.255.255.0"/>
   < /Ethernet>
```

Automatic IP configuration using DHCP server:

```
<Ethernet>
   <IP _="DHCP"/>
   <HostName _="FPGateway"/>
</Ethernet>
```

3.15 WLAN / WiFi

All FP IoT Gateways equipped with an USB host port support an FP WLAN / WiFi module (FP order number 90.0072.8100.00). The WiFi module can be configured as an access point or as a WLAN client. This chapter describes the WLAN / WiFi configuration.

3.15.1 WLAN access point mode

Insert the FP WLAN module gently into the USB host port. Wait a few seconds. Then press the button “WiFi On” (FP S Enguards W500 / W600) or “SD Unmount” (FP S-OTGuard hutline series) for at least 4 seconds. After about 20 seconds the “WiFi” LED (FP S Enguards W500 / W600) or “Active” LED (FP S-OTGuard hutline series) should flash briefly every second. The access point is now active.
Standard IP address of the WLAN access point: 192.168.100.1

Database path: /ISP/WLAN_AP

WLAN access point configuration	
Syntax	**[<SetConfig _="ISP" ver="v">** **<WLAN_AP>** **<SSID _="My_FP_Device" />** **<EnableOnStartup _="1" />** **<AllowedConnections _="1" />** **<Authentication _="WPA2" />** **<Password _="Secret Password" />** **<HostName _="WE550_Test" />** **</WLAN_AP>** **</SetConfig>]**
Description	Defines the access point settings of the FP IoT Gateway WLAN module.

Elements	***SSID*** Service Set Identifier of access point Default: Tixi-Devtype-serial ***EnableOnStartup*** This parameter determines wether the WLAN access point is automatically activated when the system starts. 0 = do not activate automatically (default) 1 = activate automatically ***AllowedConnections*** Defines how many concurrent client connections are allowed. Maximum: 5; Default: 1 ***Authentication*** Specifies the encryption method. WPA2 = WPA2 TKIP encryption (default) WPA2_CCMP = WPA2 AES encryption ***Password*** WLAN password (only ASCII characters are supported, no special characters allowed). Default: berlin2000. ***HostName*** WLAN host name (as alternative to IP address). Default: Tixi-Devtype-serial
Example	WLAN access point mode using WPA2_CCMP, custom host name, password, SSID.

```
[<SetConfig _="ISP" ver="v">
<WLAN_AP>
    <SSID _="My_own_device" />
    <EnableOnStartup _="1" />
    <AllowedConnections _="2" />
    <Authentication _="WPA2_CCMP" />
    <Password _="123456" />
    <HostName _="FPGW" />
</WLAN_AP>
</SetConfig>]
```

Automatic connection via WPS (WiFi Protected Setup)

The WiFi Protected Setup (WPS) option is supported in Access Point mode only.

WPS allows automatic connection to an access point without entering a password. To switch to WPS mode, the access point mode must already be active.

- Press the “WiFi” or “SD Unmount” button for about 1 second
- Hold down the button and press the “Service” button at the same time

The WiFi On” or “Active” LED flashes rapidly. WPS mode is now active for 2 minutes. You can now use your end device (laptop, smartphone etc.) to establish a connection with the FP device. Many end devices detect the WPS mode automatically (e.g. Windows 10) and can connect directly to the device.

After 2 minutes the FP device switches back to normal “Access Point” mode. WPS can be reactivated at any time.

3.15.2 WLAN client mode

The WiFi client mode must be configured. After a factory reset the “Accces Point” mode is initially active. Only one WLAN profile is currently supported. WLAN client mode and WLAN access point mode can’t be used simultaneously.

Database path: /ISP/WLAN

WLAN client configuration

Syntax

```
[<SetConfig _="ISP" ver="v">
<WLAN>
   <Profile_0 SSID="acer">
      <Authentication _="WPA_TKIP"/>
      <Password _="87654321"/>
      <Ethernet>
         <IP _="DHCP"/>
         <HostName _="myDeviceName"/>
      </Ethernet>
   </Profile_0>
</WLAN>
</SetConfig>]
```

Description

Defines the client mode of the FP IoT Gateway WLAN module.

Elements

SSID

Name of the access point to which the device should connect (only ASCII characters allowed).

Authentication

Specifies the encryption method.

WPA2_TKIP	= WPA2 TKIP encryption (default)

Password

WLAN password of the router. Only ASCII characters are supported, no special characters allowed.

IP

DHCP (automatic assignment of IP address, gateway and DNS by the router).

HostName

Host name by which the device can be reached on the network (if supported by the router).

Example

WLAN client mode using WPA2_TKIP, custom host name and password.

```
[<SetConfig _="ISP" ver="v">
<WLAN>
   <Profile_0 SSID="acer">
         <Authentication _="WPA_TKIP"/>
         <Password _="123456"/>
         <Ethernet>
            <IP _="DHCP"/>
            <HostName _="myDeviceName"/>
         </Ethernet>
   </Profile_0>
</WLAN>
</SetConfig>]
```

WLAN Scan command

In WiFi client mode it is possible to scan the WLAN / WiFi network for available access points.

ScanWLAN command	
Syntax	**`[<ScanWLAN ver="v">]`**
Description	Scans the WLAN / WiFi network for available access points.

Example	Result of a ScanWLAN command.
	`[<ScanWLAN>`
	`<AP0 _="" RSSI="-75">`
	`    <BSSID _="7c:dd:90:a0:c2:04"/>`
	`    <SSID _="AVM_7390"/>`
	`    <Channel _="1"/>`
	`    <Security _="WPA2"/>`
	`</AP0>`
	`<AP1 _="" RSSI="-69">`
	`    <BSSID _="7c:dd:90:a0:e2:f4"/>`
	`    <SSID _="PMS"/>`
	`    <Channel _="1"/>`
	`    <Security _="WPA2"/>`
	`</AP1>`
	`</ScanWLAN>]`

3.16 TiXML/IP

The TiXML protocol can be used via IP protocol. The appropriate TCP port and communication timeout is configured in the ISP database.

Database path: /ISP/TiXML

TiXML/IP configuration	

Syntax	**`<TiXML>`**
	`    <Port _="TCP port">`
	`    <Timeout _="delay">`
	`    <RestrictionStartIP _="Start IP">`
	`    <RestrictionEndIP _="End IP">`
	`</TiXML>`

Description	Defines the TiXML/IP TCP port and communication timeout.
Elements	*TCP port* TCP port used for communication (default 8300). *Timeout* Timeout after which the FP Gateway will close the TCP socket. *Start IP* (Optional) start IP address of an address range which is allowed to connect to the TiXML configuration port. Must be used together with End IP. *End IP* (Optional) end IP address of an address range which is allowed to connect to the TiXML configuration port. Must be used together with Start IP. Can be used to completely disable TiXML services by setting Start IP and End IP to 127.0.0.1
	TiXML/IP communication via TCP port 8300 with a socket timeout of 15 minutes.

Example

```
<TiXML>
 <Port _="8300">
 <Timeout _="15m">
</TiXML>
```

3.17 Serial Gateway

The Serial Gateway Feature is incorporated in FP IoT Gateways (HE series). Basically, it allows for IP data (being sent via Ethernet connection) to be translated into serial data. This translation enables serial-interface-only devices to be controlled remotely via any IP connection, even if such device got no IP interface of its own.

Using this feature is rather simple and requires these preconditions:

- The FP IoT Gateway must be accessible via IP connection, i.e. there must be an Ethernet connection and the FP IoT Gateway must be assigned an IP address.
- The FP IoT Gateway must be connected by serial interface to the device that is to be remotely controlled.
- The FP IoT Gateway must be configured by means of a special section within the ISP database.

This configuration is described as follows.

Database path: /ISP/SerialGateway

Serial Gateway	

Syntax	`[<SetConfig _="ISP" ver="y">`
	`    <SerialGateway>`
	`    <VirtualCOMServer_1>`
	`    <Port _="`***`Port`***`"/>`
	`    <MinSendBlockSize _="`***`MinSendBlockSize`***`"/>`
	`    <MinReceiveBlockSize _="`***`MinReceiveBlockSize`***`"/>`
	`    <CharTimeout _="`***`CharTimeout`***`"/>`
	`    <ReceiveTimeout _="`***`ReceiveTimeout`***`"/>`
	`    <ControlMode _="`***`ControlMode`***`"/>`
	`    <InactivityTimeout _="`***`InactivityTimeout`***`"/>`
	`    <COMPort _="`***`COMPort`***`"/>`
	`    <Behaviour _="`***`Behaviour`***`"/>`
	`    <ConnectTimeout _="`***`ConnectTimeout`***`"/>`
	`    <COMSettings _="`***`COMSettings`***`">`
	`        <Baudrate _="`***`Baudrate`***`"/>`
	`        <Format _="`***`Format`***`"/>`
	`        <Handshake _="`***`Handshake`***`"/>`
	`      </COMSettings>`
	`    </VirtualCOMServer_1>`
	`    </SerialGateway>`
	`</SetConfig>]`

Description	Attribute group which defines the properties of the serial-to-IP gateway of the FP IoT gateway.

Elements	***Port:*** IP port of the FP IoT Gateway that accepts the data to be converted. Note that this must not be the same port that is used to send control commands to the FP IoT Gateway. (default: 23, range: 0...65535, no unit) ***MinSendBlockSize:*** This optional parameter determines the minimum number of characters that the serial interface is to receive before an IP data packet is sent out. Note that this behaviour may be interfered with by the effects of the ***CharTimeout*** setting, described as below. (default: 1500, range: 0...1500, no unit) ***MinReceiveBlockSize:*** This optional parameter determines the minimum number of characters that the IP interface is to receive before being transmitted via the serial interface. Note that this behaviour may be interfered with by the effects of the ***CharTimeout*** setting, described as below. (default: 1500, range: 0...1500, no unit) ***CharTimeout:*** This optional parameter defines the time that is allowed to pass between two subsequent characters coming from the serial port. If the timeout is exceeded, the accumulated characters will be sent out as an IP packet, regardless if the MinSendBlockSize is hit or not. (default: 100ms, range: 0ms...1073741823ms 0s...1073740s 0m...17894m 0h...297h) ***ReceiveTimeout:*** This parameter defines how long the device waits for the number of character defined in the *MinReceiveBlockSize* parameter. (default: 100ms, range: 0ms...1073741823ms 0s...1073740s 0m...17894m 0h...297h) ***COMPort:*** This determines which COM port is used as the serial port of the FP IoT Gateway. There is no unit and range depends on how much COM ports the specific FP IoT Gateway model comes with. ***Behaviour:*** This setting controls the behaviour of the VirtualSerialCOMPortServer. It consists of a list of comma separated options, described as below:

MessageConnect	If provided, a CONNECT message is sent to the IP port once the serial port is available. If omitted (which is the default), no such message is sent once the COM port is accessible.
NoNVTNeg	If provided, negotiation of Telnet options is suppressed; however, negotiations initiated by virtual serial ports may be accepted, depending on certain circumstances. If omitted, even Telnet connections are negotiated. This option only applies if the *ControlMode* is set to "NVT", as described below.

ConnectTimeout:

This sets the time the IP port waits for the serial port to open, before the session is dropped.

(default: 100ms, range:
0ms...1073741823ms
0s...1073740s
0m...17894m
0h...297h)

ControlMode:

This parameter sets the operating mode for the VirtualCOMServer:

NVT	The VirtualCOMServer operates as a network virtual terminal and COM port settings can be set dynamically by the virtual serial port. Furthermore, device and linestate changes can be transmitted.
none	COM port settings are used as onfigured and TELNET commands regarding these settings will be ignored.

InactivityTimeout:

Defines the time that needs to pass between transmission of two subsequent characters before the VirtualCOMServer closes connection to the virtual serial port.

COMSettings:

This (optional) parameter defines the configuration mode of the COM port.

dynamic	The default value causes the COM port to be initialized with the settings provided within this group. These settings may be overwritten by the virtual serial port with other settings.
fix	The COM port will be initialized with the settings provided within this group. These settings can not be overwritten by the virtual serial port with other settings.

Baudrate:

This just sets the baudrate used on the FP IoT Gateways COM port in bits per second (bps resp. baud).

(default: 115200, range:
300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 115200)

Format:

Sets the data format on the COM port. Syntax is the same as for any COM port config and is composed of three elements: ***DatabitsParitybitStopbits***, which are explained as follows:

Databits: **8** (8 data bits), **7** (7 data bits)
Paritybit: **N** (none), **E** (even), **O** (odd)
Stopbits: **1** (one stopbit), **2** (two stopbits)

Default is **8N1**.

Handshake:

This optional parameter determines the handshake type, which by default is "none":

None	no handshake (default)
XONXOFF	Software handshake
XONXOFFPASS	Software handshake, XONXOFF being forwarded to application
RTSCTS	Hardware handshake with RTS/CTS
DTRDSR	Hardware handshake with DTR/DSR
HALF	half duplex RS485 communication
FULL	full duplex RS485 communication
HALFX	half duplex RS485 communication with XON/XOFF
FULLX	full duplex RS485 communication with XON/XOFF
noDTR	deactivates DTR
RTSDTRPower	mutual activation of RTS and DTR for bus power supply

Example

```
[<SetConfig _="ISP" ver="y">
   <SerialGateway>
      <VirtualCOMServer_1>
         <Port _="23" />
         <MinSendBlockSize _="100" />
         <CharTimeout _="10ms" />
         <MinReceiveBlockSize _="100" />
         <ReceiveTimeout _="100ms" />
         <ControlMode _="none"/>
         <InactivityTimeout _="1m"/>
         <COMPort _="COM2" />
         <Behaviour _="" />
         <ConnectTimeout _="5s" />
         <COMSettings _="fix">
            <Baudrate _="115200"/>
            <Format _="8E1"/>
            <Handshake _="RTSCTS"/>
         </COMSettings>
      </VirtualCOMServer_1>
   </SerialGateway>
</SetConfig>]
```

3.18 Webserver, PPP-Server, TFTP-Server

Information about the Webserver, PPP-Server and TFTP-Server configuration can be found in the "Webserver TiXML manual" (code WEB-EN).

3.19 Port forwarding

FP IoT Gateways Hx600 and Wx500 / Wx600 supports port forwarding from a LAN host to an internet host. Please note that at the time of writing this feature is still in beta testing (requires firmware 5.2.6.38 and newer).

The port forwarding feature allows a LAN device which is connected to the LAN interface of the FP IoT gateway to open a TCP connection to any internet host. You can define up to 3 forward rules.

This configuration is described as follows.

Database path: /ISP/Routing

Routing

Syntax

```
[<SetConfig _="ISP" ver="y">
<Routing>
  <IP_Forwarding _="1"/>
  <Enable_NAT _="1" Interface="ForwardInterface" />
  <WANInterface _="WANInterfaceID" />
  <LocalInterface _="LocalInterfaceID" />
  <Forward1 _="1" fromPort="FromPortNo"
    ToHost="ToHostAddr" ToPort="ToPortNo"
    Direction="Direction"/>
  <Forward2 _="1" fromPort="FromPortNo"
    ToHost="ToHostAddr" ToPort="ToPortNo"
    Direction="Direction"/>
  <Forward3 _="1" fromPort="FromPortNo"
    ToHost="ToHostAddr" ToPort="ToPortNo"
    Direction="Direction"/>
</Routing>
</SetConfig>]
```

Description

Attribute group which defines the properties of the port forwarding feature.

Elements

ForwardInterfaceID / WANInterfaceID / LocalInterfaceID:

Defines the local and WAN interfaces from/to which the port forwarding should be configured.

Possible interface types are (if applicable):

Interface types	Description
eth0	Local LAN interface
ppp0	Mobile connection interface (requires a 2G, 3G or 4G modem)
wlan0	WiFi connection interface (requires optional WiFi stick)
tun0	OpenVPN tunnel interface (requires an OpenVPN connection)

Forward1, Forward2, Forward3: define up to 3 different port forwarding rules.

ForwardInterfaceID should be the same like WANInterfaceID.

FromPortNo:

Defines the port for the incoming IP packets. Range: 1 .. 32767

Please use the port number with care.

ToPortNo:

Defines the port for the outgoing IP packets. Range: 1 .. 32767

Please use the port number with care.

ToHostAddr:

Specifies the host IP address (IPv4) to which the IP packets should be forwarded.

Direction:

Specifies the direction of the IP packet forwarding.

Direction	Description
In	Forward incoming IP packets (FromPort) to local host using ToPort / ToHost
Out	Forward local IP packets to internet host (outgoing) using ToPort / ToHost

Example

```
[<SetConfig _="ISP" ver="y">
<Routing>
  <IP_Forwarding _="1"/>
  <Enable_NAT _="1" Interface="eth0" />
  <WANInterface _="ppp0" />
  <LocalInterface _="eth0" />

  <Forward1 _="1" fromPort="23830" ToHost="192.168.1.23"
    ToPort="8300" Direction="In" />
  <Forward2 _="1" fromPort="2380" ToHost="195.10.80.5"
    ToPort="80" Direction="Out" />
  <Forward3 _="1" fromPort="24502" ToHost="192.168.1.25"
   ToPort="502" Direction="In" />
</Routing>
</SetConfig>]
```

Detailed description of the example configuration:

Forward1 configures port forwarding of incoming IP packets from ppp0 interface (mobile connection) on port 23830 to local LAN interface to host 192.168.1.23 on port 8300.

Forward2 configures outgoing port forwarding of IP packets coming from local LAN interface on port 2380 which will be forwarded to ppp0 interface to internet host 195.10.80.5 on port 80.

Forward3 configures port forwarding of incoming IP packets from ppp0 interface (mobile connection) on port 24502 to local LAN interface to host 192.168.1.25 on port 502.

4. Data Logging

Any operation of the FP Gateway may be logged for later review of what actually happened. Besides the system logging, a logging of process data (e.g. I/Os or PLC variables) is possible.

A maximum of 12 logfiles can be created in order not to store all data in the same place.

The LOG database therefore features two sections: LogDefinition and EventLogging. The first creates the logfiles and data structures and must contain info on their names, size and type of content. The second assigns event types to logfile names so that info regarding a specified event type will be written into a specific logfile.

The logfiles themselves are organized as ring buffers with a user defined size. If a logfile is full, the logging starts overwriting the oldest logfile entries (FIFO).

Two different types of logfiles are supported:

- XML-logfiles for xml-formatted logging (easy to read)
- Binary logfiles (less memory usage)

4.1 LogDefinition

The "LogDefinition" contains the "LogFiles" and the "Records" group inside LOG database.
Both groups are described in the following chapters.

Database path: /LOG/LogDefinition

```
<LogDefinition>
   <LogFiles>
      <Event size="10240"/>
      <JobReport size="10240"/>
      <Datalogging_0 size="20480" contenttype="binary"
         record="Datalogging_0"/>
   </LogFiles>
   <Records>
      <Datalogging_0>
         <Variable_0 _="int" size="2"/>
         <Variable_1 _="Uint16"
               path="/Process/Bus1/Device_0/Variable_1"/>
      </Datalogging_0>
   </Records>
</LogDefinition>
```

Note:
Projects created by TILA2 still use an older log database format, where the LogFiles and Records are stored within separated groups, but with same content:
- /LOG/LogFiles
- /LOG/Records

The Group "LogDefinition" does not exist.

4.1.1 LogFiles Group

LogfFiles	
Syntax	**`<LogFileName size="LogFileSize"contenttype="Type" record="RecordPath"/>`**
Description	Defines logfiles by their name and size.
Elements	*LogFileName* The identifier of the logfile. Random unique names may be chosen. A maximum of 12 logfiles can be created. *LogFileSize* Specifies the logfile size in bytes. A logfile size bigger than the FP Gateway memory will be rejected with an error message. Because of the file system structure, the logfile size will be round up to 512byte sectors. *Type* The content type for logfiles has to be set to "xml" for XML formatted data logging and to "binary" for binary data logging. *RecordPath* The record path refers to a record inside "Records" database which defines the data structure of a binary log entry.

Example	Create XML-logfile *Log1* with 1KB size and binary logfile *Log2* with 20KB size and 80 bytes maximum for an entry:
	`<LogFiles>`
	`   <Log1 size="1024"/>`
	`   <Log2 size="20480" contenttype="binary" record="Struct"/>`
	`</LogFiles>`

During upload of logfile definitions, the number of currently existing logfiles plus the number of the uploaded logfiles must not exceed twelve, because new logfiles are written before existing logfiles will be deleted. In this case, it will be necessary to do a factory reset before uploading the new logfile definition.

4.1.1.1 SupportLog

After a factory reset, the FP Gateway is preconfigured with a Logfile "SupportLog" which may also be added to TiXML projects.
This logfile is used to collect information about the incorporated PSTN or GSM device like country setting, SMSC, own numbers etc. during system startup.

4.1.2 Records Group

Records	

Syntax	***`<RecordName>`***
	`   <ValueName _="Type"/>`
	`   <ValueName _="Type" size="Length"/>`
	`   <ValueName _="Type" size="Length" value="Value"/>`
	`   <ValueName _="Type" size="Length"`
	`    format="FormatString"/>`

```
   <ValueName _="Type" path="Source"/>
   <ValueName _="Type" path="Source" exp="Exponent"/>
   <ValueName _="Type" path="Source"
    multip="Factor+Offset"/>
   <ValueName _="Type" path="Source" Name="Alias"/>
</RecordName>
```

Description	Defines the structure of a binary logfile.
Elements	*RecordName* Name of the data structure the logfile refers to *ValueName* Name of the value being assigned during logging *Type* Type of value: int: integer string: text string byte: byte value word: word (16bit) value dword: dword (32bit) value float: float (32bit) value double: double (64bit) value meterbus: Meterbus RAW data (only usable with "path", not "value"!) Additional simpleTypes, see 6.5.1 Bit: bit Int8: byte (8bit) signed Uint8: byte (8bit) unsigned Int16: word (16bit) signed Uint16: word (16bit) unsigned Int32: dword (32bit) signed Uint32: dword (32bit) unsigned *Length* Number of bytes registered for each log entry (max 100). Must be defined for type "string", optional for all other types. int: max. 4 bytes, default 4 bytes string: max. 100 chars, no default byte: default unsigned (1 byte) word: default unsigned (2 byte) dword: default unsigned (4 byte) float: default 4 byte double: default 8 byte meterbus: size will be calculated during logging Bit: bit (1/8 byte) Int8: default signed (1 byte) Uint8: unsigned (1 byte) Int16: signed (2 byte) Uint16: unsigned (2 byte) Int32: signed (4 byte) Uint32: unsigned (4 byte)

Value

Value for the log entry, if not specified in the log command (option). May be given via reference.

Exponent

Exponent of base 10 to specify fix point precision of
simpleType = Uint8, Uint16, Uint32, Int8, Int16, Int32 (see chapter 6.5.1).
The logged variable value will be multiplied by 10^{Exp} to get the output value.
Output value = 10^{Exp} * logged variable value.

The exponent therefore specifies the position of comma within a fix point value.

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

Factor+Offset

The logged value will be multiplied by this factor and the offset will be added to get the output value.
Only simpleType = Uint8, Uint16, Uint32, Int8, Int16, Int32 (see chapter 6.5.1).

Output value = Factor * Logged value + Offset

The factor is used as a fraction, e.g.: "1/1000" or "3600/1", the denominator and numerator must not be zero. The offset may be negative or positive.

Source

Source for the log entry value, if not specified in the log command (option). Will be processed faster than "value" but supports external (PLC) variables and some system properties (e.g. time values, serial number) only.

FormatString

String that defines the logfile value output format.

	For a list of available format options, see chapter 6.5.2. *Alias* Replaces the ***ValueName*** in the header of non-XML logfile output by the given alias name. If the ReadLog / IncludeLog(TXT) flag "UseAlias" is used, an attribute "Name" with the variable alias will be added on XML output, too.

Example	This example creates a data structure with *4 values*:
	`<Records>`
	`  <Datalogging_0>`
	`    <Value1 _="Int16" Name="word variable"/>`
	`    <Value2 _="Int8" value="&#xae;/Process/C40/IB/P0;"`
	`      Name="byte variable"/>`
	`    <Value3 _="Bit" value="&#xae;/Process/C40/I/P8;"`
	`      format="?On,Off;" Name="bit variable"/>`
	`    <Value4 _="word" path="/Process/MB/A/AI/P0"`
	`    multip="1/10+3"`
	`      Name="word variable 2"/>`
	`  </Datalogging_0>`
	`</Records>`
	
	Value1 may be a Int16 signed 16bit word value (2 byte) with a value range of -32768 to 32767. The value must be given by the event.
	Value2 is an Int8 signed byte value (1 byte) with a value range of -128 to 127. The input ports P0-P7 (via reference) are automatically written as this byte with every log process.
	Value3 is a bit value. The input port P8 is automatically written as this bit. Instead of the 0/1 value the string "On" or "Off" will be output during reading the logfile.
	Value4 is the analog input value devided by 10 with an offset of +3.
	The head line of the logfile output will show the variable alias names.

4.2 EventLogging

Database path: /LOG/EventLogging

EventLogging Database	
Syntax	**`<EventSource mode="Mode1Mode2" file="LogFileName"/>`**
Description	Defines what system information to write into which logfile.
Elements	*EventSource* The identifier for the kind of event to log into the given logfile. Possible EventSource Values are as follows: **`Event`** — reports all events that become processed **`Login`** — reports all cases of anyone doing or attempting a login into the FP Gateway as well as for logout **`IncomingMessage`** — reports all incoming messages **`FailedIncomingCall`** — reports all incoming calls that could not be handled properly **`JobReport`** — reports the result of sending a message, regardless if it was ok or not. *Mode1*

Specifies which incidents to report into the logfile. Possible values:

a	report all incidents (errors and ok)
e	report errors only
o	report ok notifications only

Mode2

Specifies the verbosity of the logfile entry. Possible values are as follows:

v	verbose messages telling the possible reason
[empty]	giving a short description only **(default)**

LogFileName

The name identifier of the logfile. Choose one of the Logfiles of the LogDefinition (see chapter 4.1.1).

Example

Verbosely report of failed message sending to Log1 and - shorter - all events triggered to Log2:

```
<EventLogging>
   <JobReport mode="ev" file="JobReport"/>
   <Event mode="a" file="Event"/>
</EventLogging>
```

4.3 Logging commands

Log Command

Syntax

```
<Log _="LogfileName">
   LogData
</Log>
```

Description

Creates an entry with following structure:

```
<ID_nnn time=_"TimeStamp">
   LogData
</ID_nnn>
```

nnn:	consecutive number to address the log entry
TimeStamp:	System time when the log entry was written

Note:
The Log command can only be used for logfiles defined with the content type XML.

Elements

LogfileName:
Name of the logfile where the data is written to. Must be defined in the LogDefinition database.

LogData:
Data to be logged. May be collected via references.

Example

Event handler that logs the last power off and the last power on time.

```
<PowerOn >
   <Log _="SupportLog">
      <PowerOff _="&#xae;/TIMES/PowerOffTime;"/>
       <PowerOn _="&#xae;/TIMES/PowerOnTime;"/>
   </Log>
</PowerOn>
```
Result:
```
<ID_1 _="2003/08/13,10:31:55">
   <PowerOff _="2003/08/13,09:10:00" />
   <PowerOn _="2003/08/13,09:16:52" />
</ID_1>
```

BinLog Command	
Syntax	***<BinLog _="LogfileName">*** ***<ValueName _="Value"/>*** ***<ValueName _="Value"/>*** ***. . .*** ***</BinLog>***
Description	Creates a binary logfile entry with the structure of the given record. Note: The BinLog command can only be used for logfiles defined with the content type "binary" and an assigned record.
Elements	*LogfileName:* Name of the logfile where the data is written to. Must be defined in the LogDefinition database. *ValueName:* Name of value defined in record database. Only necessary if no value or path is defined within the record. Value: Value to be written into the structure. Only necessary if no value or path is defined within the record.

Example	Value1 is given as "Parameter" during DoOn. Value2 and Value3 are port values given by the record value definition.
	
	`<LogValues>`
	`   <BinLog _="Datalogging_1">`
	`      <Value1 _="&#xae;~/Parameter"/>`
	`   </BinLog>`
	`</LogValues>`
	
	The logfile entry with Parameter=12345 may look like this:
	
	`<ID_1 _="2003/08/21,09:58:59">`
	`   <Value1 _="12345"/>`
	`   <Value2 _="32"/>`
	`   <Value3 _="On"/>`
	`</ID_1>`

4.4 Logfile memory calculation

This information may be used to calculate the necessary logfile memory depending on the log interval and amount of data.

Type		size	amount
Logfile Header		56 Byte	per file
Entry Header		12 Byte	per entry
Entry Data	XML	Size of XML text (variable)	per entry
	Binary	Sum of data elements (static)	per entry

Example

Log periode:	1 week
Log cycle:	10 minutes

Record:

```
<Datalogging_0>
   <Value1 _="int" size="2"/>
   <Value2 _="int" size="2"/>
   <Value3 _="int" size="2"/>
   <Value4 _="int" size="2"/>
   <Value5 _="int" size="2"/>
   <Value6 _="int" size="2"/>
   <Value7 _="int" size="2"/>
   <Value8 _="int" size="4"/>
   <Value9 _="int" size="1"/>
   <Value10 _="int" size="1"/>
  </Datalogging_0>
```

Calculation:
(20 Byte Data + 12 Byte Header) * 144 entries/day * 7 days + 56 Bytes File Header = 32312 Bytes
Logfile size should be 32768 (divisible by sector size 512 Byte)

Estimated memory usage with 2MB memory (only 1 MB available during sending!):

Value to log	Log-Sessions	With 1 minute interval overwrite after	With 15 minute interval overwrite after	With 1 hour interval overwrite after
1 Byte	77000	53 days	26 month	9 years
1 DWord	62500	43 days	21 month	7 years
10 Byte	45500	31 days	15 month	5 years
10 DWord	19200	10 days	5 month	2 years

4.5 Reading and clearing logfiles

For information on how to read or clear the content of logfiles see chapter 2.4.6.6.

4.6 Sending and formatting log reports

The FP Gateway is able to include logged data into messages. You can choose two different commands to include logdata into messages:

1. IncludeLog. The logged data will be included in XML format.
2. IncludeLogTXT: Most applications can’t handle XML so we’ve implemented a feature to format the output of the logged data. You can choose the predefined logfile formats "CSV", "HTML" and "XML" or reformat the data at your own wish.

IncludeLog – include entries from the log files into message text	
Syntax	**`<IncludeLog _="`*`LogFileName`*`" range="`*`entryrange`*`"/>`**
Description	Template processor instruction which includes logfile entries in the text of a message. The name of the logfile can be specified as well as a range of entries to be inserted. The generated output is similar to the output generated by the ReadLog command. See chapter 2.4.6.6 for details.
Parameter	*LogFileName:* Name of the logfile to read. *entryrange:* Range of log data to send. See chapter 2.4.6.6 for details.

Example	Send the entries with the IDs 7 – 8.
	`<UserTemplates>`
	`   <LogfileMsg>`
	`      <IncludeLog _="Datalogging_0" range="ID_7-ID_8"/>]`
	`   </LogfileMsg>`
	`</UserTemplates>`
	Message Body result:
	`<ID_7 _="2002/10/10,16:10:51">`
	`   <Data _="Logged Data"/>`
	`   </ID_7>`
	`   <ID_8 _="2002/10/10,16:10:51">`
	`   <Data _="Logged Data"/>`
	`</ID_8>`

IncludeLogTXT – include and reformat entries from the log files into message text	
Syntax	**`<IncludeLogTXT _="`*`LogFileName`*`" range="`*`entryrange`*`" type="`*`templates`*`" flags="`*`header`*`" fillInterval="`*`interval`*`" maxInterval="`*`tolerance`*`" fillText="`*`string`*`" Viewset="variables" `*`Formats`*`/>`**
Description	Template processor instruction which includes logfile entries in the text of a message or into an attachment. The name of the logfile can be specified as well as a range of entries to be inserted. The generated output depends on the specified format parameters.
Parameter	*LogFileName:*

	Name of the logfile to be included in the message text. *entryrange:* Range of log data to send. See chapter 2.4.6.6 for details. *templates:* Predefined logfile formats: *CSV*: "character separated values", e.g. for easy Excel import. *HTML:* Logdata will be formatted as HTML table *XML:* Logfile will be send as XML file *header* flags="NoId,NoDate,NoTime,NoNames,NoSec,UseAlias,CRC16,CRC32" *NoId:* removes the ID of each entry (only for none XML structures) *NoDate:* removes the Date of each entry (only for none XML structures) *NoTime:* removes the Time of each entry (only for none XML structures) *NoNames:* removes the first row with variable names (only for none XML structures) *NoSec:* removes the seconds within the time stamp *UseAlias:* adds the variable alias names to the XML logfile output *CRC16:* calculates a CRC16 checksum (decimal) over the logfile output (excluding headers) and writes it under the data (only for CSV). CRC16 calculation: Width: 16Bit Polynomal: 0x1021 Init Value: 0x0000 Reflection: In/Out deactivated XOR Out: 0x0000 *CRC32:* calculates a CRC32 checksum (decimal) over the logfile output (excluding headers) and writes it under the data (only for CSV). CRC32 calculation: Width: 32Bit Polynomal: 0x04C11DB7 Init Value: 0xFFFFFFFF Reflection: In/Out deactivated XOR Out: 0x00000000 *interval:* Expected log interval. If the time between two log entries exceeds the ***tolerance*** interval, an entry with the content of ***string*** will be added with the timestamp of the last entry + ***interval***. Can be used to create a fixed log content length if the FP Gateway was switched off or the logging was stopped for a while. *tolerance:* Maximum time between two log entries before ***string*** will be added. *string:* String added to the log output if ***tolerance*** interval was exceeded (only for CSV format). *variables:*

	List of variables (separated by comma) to be selected for logfile output. The variable names must match the tag names of the record entries.

Formats:

tabstart	string which will be added at the beginning of the file maximum length: 30 chars, default: (empty)
tabend	string which will be added to the end of the file maximum length: 30 chars, default: (empty)
tagstart	string which will be added in front of each value. maximum length: 30 chars, default: "
tagend	string which will be added to the end of each value. maximum length: 30 chars, default: "
colsep	string which will be added between the values (between tagend and tagstart). maximum length: 30 chars, default: ;
rowstart	string which will be added in front of the first value (in front of tagstart). maximum length: 30 chars, default: (empty)
rowend	string which will be added to the end of the last value (after tagend). maximum length: 30 chars, default: (CRLF)
rowsep	string which will be added between two rows (in front of rowstart). maximum length: 30 chars, default: (empty)
cols	defines the number of columns after which a line break (rowend/rowstart) will be made. Useful to distribute devices to several lines of a CSV file. *auto* number of columns equal number of record entries (default) *0…32768* number of columns
crcText	string added in front of the CRC
Attention:	If you enter a rowend character, the default CRLF will be replaced. To get each log entry in a seperate line, you'll have to add the CRLF (&#x0a;&#x0d;) manually to the end of the rowend character, e.g. if you like to get an exclamation mark as rowend, enter this: `rowend="!&#x0d;&#x0a;"`

Example

Logged Data:

```
<ID_7 _="2002/10/10,16:10:00">
   <Data1 _="Logged Data1"/>
   <Data2 _="Logged Data2"/>
   <Data3 _="Logged Data3"/>
</ID_7>
<ID_8 _="2002/10/10,16:20:03">
   <Data1 _="Logged Data1"/>
   <Data2 _="Logged Data2"/>
   <Data3 _="Logged Data3"/>
</ID_8>
```

Example 1:

Send the entries with the IDs 7 – 8, remove ID and use CSV format:

```
<UserTemplates>
   <Message_0>
      <IncludeLogTXT _="Datalogging_0" range="ID_7-ID_8"
       flags="NoId" type="CSV"/>
   </Message_0>
</UserTemplates>
```

Message Body result:

```
Date;Time;Data1;Data2;Data3
2002/10/10;16:10:00;Logged Data1;Logged Data2;Logged Data3
2002/10/10;16:20:03;Logged Data1;Logged Data2;Logged Data3
```

Example 2:

Send the entries with the IDs 7 - 8, remove ID, remove date and time stamp seconds and use CSV format with comma as character separator (for english Excel):

```
<UserTemplates>
   <Message_0>
      <IncludeLogTXT _="Datalogging_0" range="ID_7-ID_8"
       flags="NoId,NoDate,NoSec" type="CSV" colsep=","/>
   </Message_0>
</UserTemplates>
```

Message Body result:

```
Date,Time,Data1,Data2,Data3
16:10,Logged Data1,Logged Data2,Logged Data3
16:20,Logged Data1,Logged Data2,Logged Data3
```

Example 3:

Send the entries with the IDs 7 – 8, remove ID, Date, Time, Names, use HTML format:

```
<UserTemplates>
   <Message_0>
      <IncludeLogTXT _="Datalogging_0" range="ID_7-ID_8"
       flags="NoId,NoDate,NoTime,NoNames" type="HTML"/>
   </Message_0>
</UserTemplates>
```

Message Body result (Web browser view):

Logged Data1	Logged Data2	Logged Data3
Logged Data1	Logged Data2	Logged Data3

HTML-code:

```
<table border=1>
   <tr>
      <td>Logged Data1</td>
      <td>Logged Data2</td>
      <td>Logged Data3</td>
   </tr>
   <tr>
      <td>Logged Data1</td>
      <td>Logged Data2</td>
      <td>Logged Data3</td>
   </tr>
</table>
```

Example 4:

Send the entry with the ID 8, remove ID, Date, Time, Names and use user defined format:

```
<UserTemplates>
   <Message_0>
      <IncludeLogTXT _="Datalogging_0" range="ID_8"
        flags="NoId,NoDate,NoTime,NoNames" tagstart="#"
        tagend="#" colsep="+" rowstart="-" rowend="-
        &#x0a;&#xod; "/>
   </Message_0>
</UserTemplates>
```

Message Body result:

```
-#Logged Data1#+#Logged Data2#+#Logged Data3#-
```

Example 5:

Send all entries, remove ID and date and create dummy entries if there are entries missing, according to the log interval of 5 minutes:

```
<UserTemplates>
   <Message_0>
      <IncludeLogTXT _="Datalogging_0" range="all"
       flags="NoId,NoDate" type="CSV" fillInterval="5m"
       maxInterval="305s" fillText="-;-;-" />
   </Message_0>
</UserTemplates>
```

Message Body result:

```
Date;Time;Data1;Data2;Data3
16:10:00;Logged Data1;Logged Data2;Logged Data3
16:15:00;-;-;-
16:20:03;Logged Data1;Logged Data2;Logged Data3
```

Example 6:

Send all entries in XML format and add variable alias names:

```
<UserTemplates>
   <Message_0>
      <IncludeLogTXT _="Datalogging_0" range="all"
        flags="UseAlias" type="XML"/>
   </Message_0>
</UserTemplates>
```

Message Body result:

```
<ID_7 _="2002/10/10,16:10:00">
   <Data1 _="Logged Data1" Name="Alias name for Data1"/>
   <Data2 _="Logged Data2" Name="Alias name for Data2"/>
   <Data3 _="Logged Data3" Name="Alias name for Data3"/>
```

```
</ID_7>
<ID_8 _="2002/10/10,16:20:03">
   <Data1 _="Logged Data1" Name="Alias name for Data1"/>
   <Data2 _="Logged Data2" Name="Alias name for Data2"/>
   <Data3 _="Logged Data3" Name="Alias name for Data3"/>
</ID_8>
```

Example 7:

Send all entries of variables "Data1" and Data3", remove ID and date and calculate a CRC32:

```
<UserTemplates>
   <Message_0>
      <IncludeLogMTXT _="Datalogging_0" range="all"
       flags="NoId,NoDate,CRC32" crcText="CRC32="
       type="CSV" Viewset="Data1,Data3"/>
   </Message_0>
</UserTemplates>
```

Message Body result:

```
Date;Time;Data1;Data3
16:10:00;Logged Data1;Logged Data3
16:20:03;Logged Data1;Logged Data3
CRC32=2481894213
```

Example 8:

Send the entries with the IDs 7 – 8, remove ID and names and use CSV format with one column:

```
<UserTemplates>
   <Message_0>
      <IncludeLogTXT _="Datalogging_0" range="ID_7-ID_8"
       flags="NoId,NoNames" type="CSV" cols="1"/>
   </Message_0>
</UserTemplates>
```

Message Body result:

```
2002/10/10;16:10:00;Logged Data1
2002/10/10;16:10:00;Logged Data2
2002/10/10;16:10:00;Logged Data3
2002/10/10;16:20:03;Logged Data1
2002/10/10;16:20:03;Logged Data2
2002/10/10;16:20:03;Logged Data3
```

4.6.1 Predefined format tags

	CSV	XML	HTML
tabstart		`<TABLE>\r\n`	`<table border=1>\r\n`
tabend		`</TABLE>\r\n`	`</table>\r\n`
tagstart		`<T v="`	`<td>`
tagend		`"/>\r\n`	`</td>\r\n`
rowstart		`<TAG>\r\n`	`<tr>\r\n`
rowend	`\r\n`	`</TAG>\r\n`	`</tr>\r\n`
colsep	`;`		

(\r = Carriage Return , \n = Line Feed)

4.6.2 Sending logfiles as attachment

You can use the IncludeLogTXT message text template within a special UserTemplates group called "Attachments". This will create an email file attachment of the logged data. The email attachment will be BASE64 coded.

The `Attachments` attribute must be added to the MessageJobTemplate:

Database path: /TEMPLATE/MessageJobTemplates

```
<MessageJobTemplates>
  <LogfileMail _="SMTP">
  <Recipient _="/D/AddressBook/TaskForce1"/>
  <Sender _="/D/AddressBook/MySelf"/>
  <Body _="/UserTemplate/LogfileMsg"/>
  <Subject _="Logfile"/>
  <Attachments _="/D/UserTemplates/Attachments/Datalogging_0"/>
</MessageJobTemplates>
```

This attachment format has to be configured in a special attachments group inside the UserTemplates database:

Database path: /TEMPLATE/UserTemplates

```
<Attachments>
  <Attachment_Set>
    <Attachment filename="file">
      <Content/>
      <Content/>
    </Attachment>
    <Attachment filename="file">
      <Content/>
      <Content/>
    </Attachment>
    ...
  </Attachment_Set>
</Attachments>
```

Example

```
<Attachments>
  <Datalogging_0>
    <Attachment filename="Datalogging_0.csv">
      <IncludeLogTXT _="Datalogging_0" range="previous 1 days"
      flags="NoId" type="CSV"/>
    </Attachment>
  <Datalogging_0>
</Attachments>
```

Some email programs, e.g. newer versions of "Outlook" or "OutlookExpress" are deleting CSV attachments automatically for security reasons. You can disable this function by changing the security settings of these programs (see email program manual).

Log data can be send as compressed email attachments, too. That reduces data by approximately 70% or more.

Example
(Transform a log file "Datalogfile_0" into a csv log file and compress it with gzip as an Attachment.)

```
<Attachments>
  <Datalogging_0>
    <Attachment filename="Datalogging_0.csv.gz" pack="gzip">
      <IncludeLogTXT _="Datalogging_0" range="all" flags="NoId" type="CSV"/>
    </Attachment>
  <Datalogging_0>
< /Attachments>
```

4.7 Logfile Counter

The FP Gateway automatically creates log counters within system properties path `/LogCounter/Logfilename` (see chapter 14).

- Each logfile entry increases the counter value by 1.
- The current value can be changed (e.g. reset to 0) by "set" command (see chapter 2.4.6.5).
- The log counter will be reset to 0 if a logfile is added/removed to the log configuration
- The log counter will be reset to 0 if the logfile size is changed

5. Remote Control

5.1 Overview

Remote control must be separated in two different connection types.

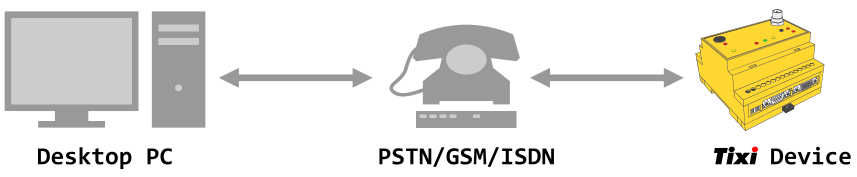
- Remote control of the FP Gateway
- Remote control of a device attached to a FP Gateway

The following chapters describe these two cases in detail. Read the chapter 'Remote Control of the FP Gateway' first because the remote control of the attached device is based on this chapter.

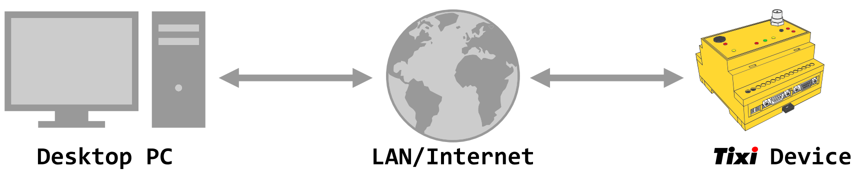
5.2 Remote Control of the FP Gateway

The following pictures show the communication ways when controlling FP Gateway by remote.

Remote control (remote configuration) via PSTN/GSM/ISDN:



Remote control (remote configuration) via LAN/Internet:

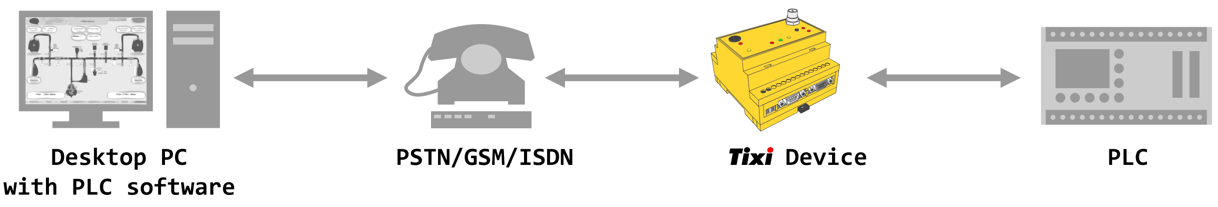


In these configurations, the desktop PC controls the remote FP Gateway by means of the TiXML protocol. To do this, the following steps are required:

1. The Desktop PC needs to *establish a connection* to the remote device. This may be a dialup connection via modem (ISDN/PSTN/GSM) using the phone number of the FP Gateway or a TCP/IP connection via LAN/WLAN, GPRS or internet using the IP address of the FP Gateway.
2. *Login* to the remote FP Gateway.
3. Control the remote FP Gateway by means of *TiXML*. (see chapter 2.4.6ff)
4. The remote connection is terminated by the *Logout* command.

5.3 Remote control of an attached PLC

The following picture shows the communication where the PLC attached to the remote Device is controlled by the desktop PC.



In this case the programming tool of the attached device (PLC, meter, etc.) is used. The connection is established in two main steps.

1. *Establish a remote control connection* to the FP Gateway as described in the previous chapter.
2. Send the *TransMode command* to switch the remote FP Gateway to the Transparent Mode (see chapter 2.4.6.1).

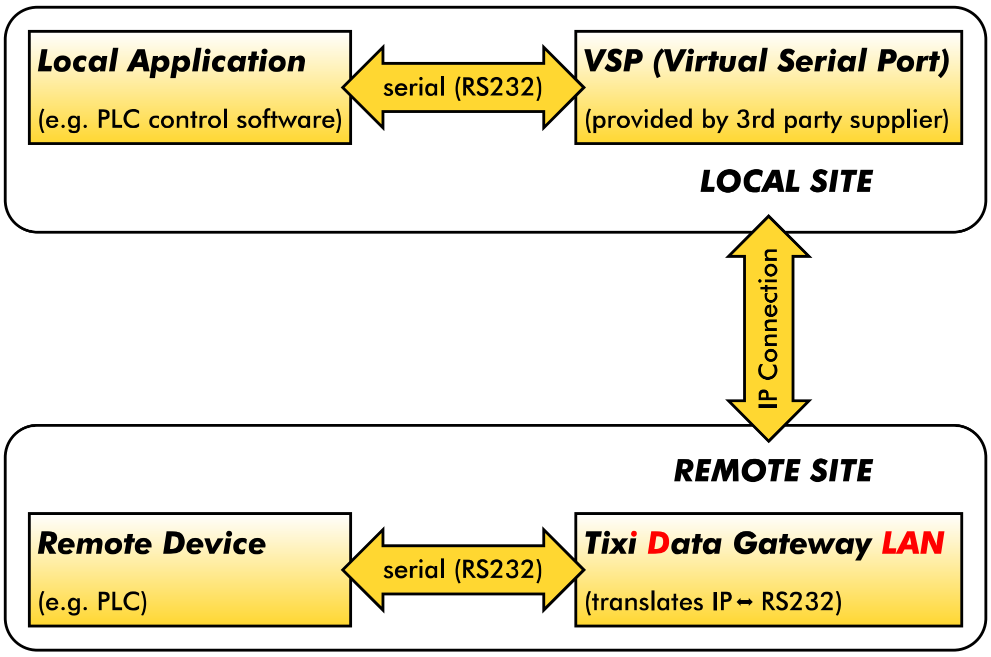
The attached device can now be controlled by the desktop PC by accessing the COM port of the dialing modem.
Such a connection can only be closed by a modem disconnection. To do this, the desktop PC sends the modem escape sequence ("+++") or the line is interrupted, e.g. by switching off the dialing modem.

5.4 Remote Control via Serial IP

Using the Serial IP feature, a remote PLC can be maintained and controlled via IP connection, just as if connected via local COM port.

Therefore, the local PLC control software connects to a Virtual Serial Port (*HW-Group VSP*), which is provided by a 3rd party supplier. This VSP converts the serial data to IP packet data, which then can be relayed over any IP connection (http, GPRS, Ethernet ...) to the remote FP IoT Gateway LAN.

This device is connected via serial RS232 to the PLC. Please see the following diagram for an outline of Serial IP operation:

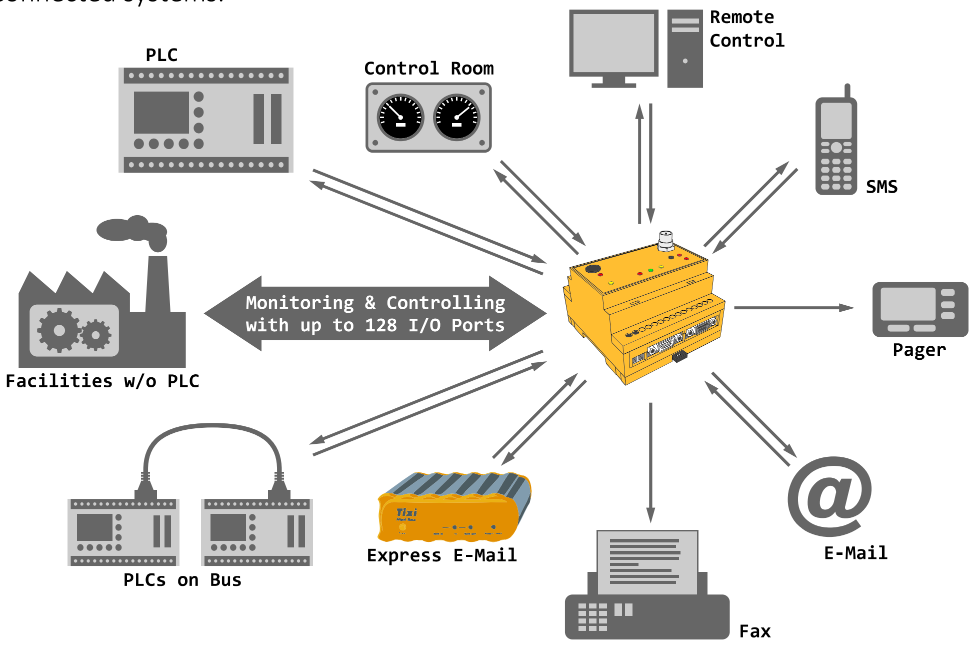


Note:
For further information to the Serial IP use the SerialIP-Manual ("Tixi.Gate LAN/GSM Hardware-Handbuch").

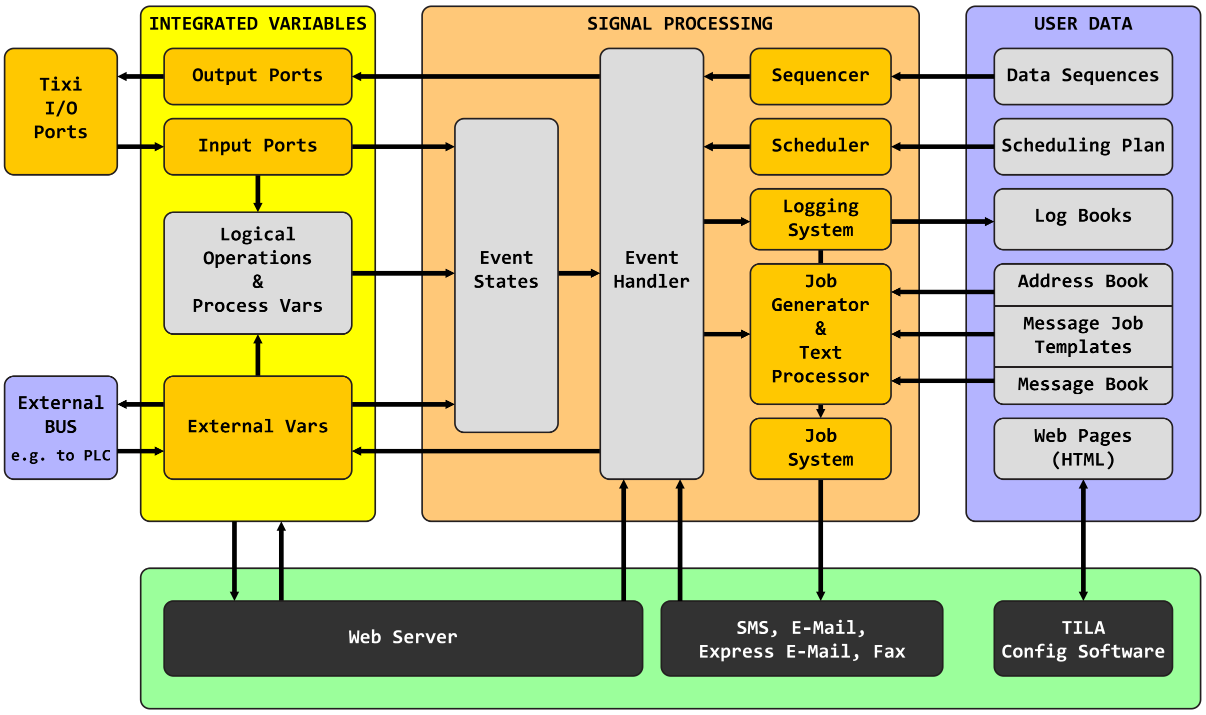
6. Process I/O Ports and Variables

6.1 Introduction

The FP Gateway can use its on board or extension I/Os as well as variables of a connected PLC to signal states of connected systems.



To handle the changes of I/Os and PLC variables, the 'Process' subsystem is part of the FP Gateway firmware. The following picture shows how it triggeres the event processing.



The process detects the changes of variables, e.g. the analog input which is mapped to the process variable path `/Process/MB/A/AI/P0`.

This leads to a processing of the EventStates which may refer to the ProcessVariable database with RPN (reverse polish notation) instructions to check event conditions, e.g. to compare the current temperature with a threshold.
The EventStates then triggers an EventHandler to process event commands, e.g. sending a message. Sending a message starts the JobGenerator which creates the jobs in the same way as if the events were received as DoOn commands from a client device (e.g. PC).

The EventStates remember whether an event was already triggered or not. Therefore, only changing of the process variable from FALSE to TRUE starts the event. To fire the same event at a later time, the process variable must change from TRUE to FALSE and then from FALSE to TRUE again.

The following sections describe the configuration of the processing in detail.

6.2 Event States

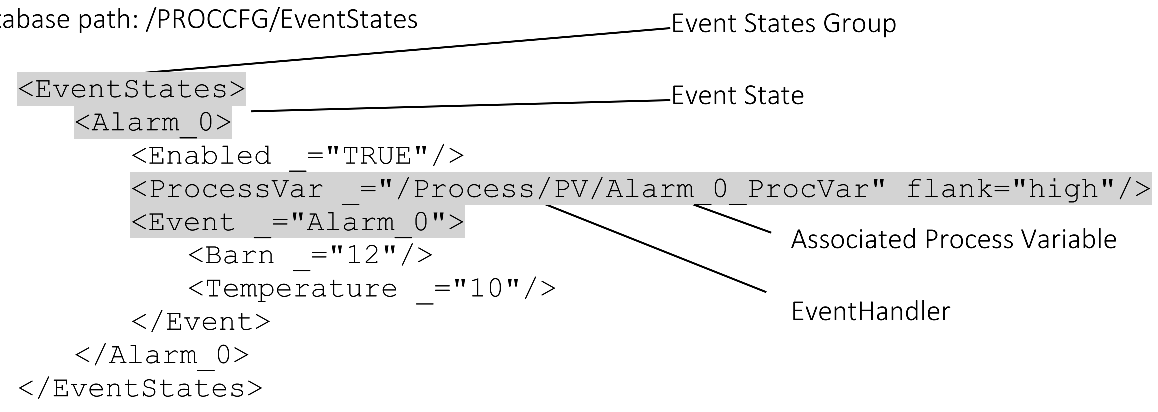
EventStates are used to trigger an EventHandler in a way that corresponds to the change in state of an associated variable. The variable may be a bit variable (e.g. I/O) or a bit result of a process variable. Variable values greater than 1 will be interpreted as 1 (TRUE).

The maximum amount of EventStates is 100.

There are two different ways to assign an event to a process variable:

- With use of the ProcessVars database (see chapter 6.2), which is slower but more flexible and allows logical operations and comparisons.
- Without use of ProcessVars database, which is much faster but doesn't support logical operations and comparisons.

Database path: /PROCCFG/EventStates



```
<EventStates>
   <Alarm_0>
      <Enabled _="TRUE"/>
      <ProcessVar _="/Process/PV/Alarm_0_ProcVar" flank="high"/>
      <Event _="Alarm_0">
         <Barn _="12"/>
         <Temperature _="10"/>
      </Event>
   </Alarm_0>
</EventStates>
```

Event State Configuration	
Syntax:	***<EventStateName>*** **<Enabled _="*EnableState*"/>** **<ProcessVar _="*ProcessPath*" *flank*="*State*"/>** **<Event _="*EventName*">** ***ParameterList*** **</Event>** ***</EventStateName>***
Description	Attribute group which defines an event state.

Elements

EventStateName:
Name of the event state, which must be unique inside this group.

EnableState:
Only checked after SetConfig or system start, no dynamic change possible.

TRUE	The event state is enabled, event will be triggered.
FALSE	The event state is disabled, event will not be triggered.
FAST	The Event is processed immediately after checking its EventState (with TRUE a list of changed EventStates will be created first).
1	same as TRUE
0	same as FALSE

ProcessPath:
Path to a variable defined in the "/Process/" tree of the system properties (see chapter 14).

State:

high	Event is triggered if associated variable changes from 0 to 1
low	Event is triggered if associated variable changes from 1 to 0
both	Event is triggered on every logical change (0 to1 or 1 to 0) of associated variable.

EventName:
Name of the triggered event if condition (process variable flank) is fulfilled.

Note:
There must be an Event Handler defined with the very same name.

ParameterList:
List of XML encoded parameters. A parameter is written in a single XML- tag with:
`<ParameterKey _="Value"/>`

where

ParameterKey	name of the Parameter (unique in the parameter list)
Value	value of the Parameter.

These parameters are passed to the Event and can be referred using &#xae;~/ParameterKey (see chapter 3.1.1).

Example

Event state configuration that triggers the EventHandler "`Alarm_0`" if the process variable `Alarm_0_ProcVar` changes from 0 to 1. Two parameters are passed to the event and can be used within its context e.g. for message text variables.

```
<Alarm_0>
   <Enabled _="TRUE"/>
   <ProcessVar _="/Process/PV/Alarm_0_ProcVar"/>
   <Event _="Alarm_0">
      <Barn _="12"/>
      <Temperature _="10"/>
   </Event>
</Alarm_0>
```

6.3 Process Variables

Typically, a system like a FP Gateway checks the state of a system. This system is called the 'process'. The process is described by 'Process Variables' representing the state of the process. Each process variable should get a meaningful name which must be unique inside the configuration.

The value of a process variable can either be set manually or calculated by reading the values of the I/O ports or PLC variables. This can be done on access or triggered by an EventHandler. For example, a signal input can be linked with another input that indicates that the process power is on, so the input signal is only valid if both conditions are met.

The general characteristics are defined in the *"PROCCFG"* database. The database contains some attribute groups inside the group ProcessVars. Each group has the name of the process variable it defines.

You may create an independent variable which may be used as a marker (memory) for integer values or strings (up to 20 characters). Up to 1000 independent variables can be used. A default value and output format is optional.

If a value source can not be resolved, an alternative value may be given, seperated by comma, e.g.:
`<LD _="/Process/Bus1/Device_0/Variable_0,10"/>`

The following example shows three different types of process variables:

Database path: /PROCCFG/ProcessVars

```
<ProcessVars>
   <Alarm_0_PV>
      <Value>
         <LDN _="/Process/MB/IO/I/P0"/>
      </Value>
   </Alarm_0_PV>

   <Counter>
      <Process>
         <LD _="/Process/PV/CounterValue"/>
         <ADD _="1"/>
         <ST _="/Process/PV/CounterValue"/>
      </Process>
   </Counter>

   <CounterValue def="0"/>
</ProcessVars>
```

Process Variable name

Value calculated on access

Calculation triggered by EventHandler "Process"

Independent variable

There are two types of process variables: DWORD and FLOAT.
Both types are working on different stacks (DWORD stack and FLOAT stack).

It is important to know that for each type are different mathematical and logical operations defined. So if you want to use a float type process variable you need to use e.g. ADDF, DIVF, LDF, POWF.

It is possible to copy / convert values form the DWORD stack to the FLOAT stack and vice versa.

Process Variable Configuration	

Syntax	***<ProcessVariableName type="PVType" exp="Exponent" format="FormatString"***
	 def="Value">
	   <Type precision="Precision">
	     RPN Instruction List
	   </Type>
	 </ProcessVariableName>

Description	Attribute group which defines a process variable. The data type of a calculated process variables is Int32 (see chapter 6.5.1)
Elements	*ProcessVariableName:* Name of the process variable. Must be unique in this database and must not be an instruction name. *PVType:* Type of the process variable (optional). If type is omitted a DWORD process variable is defined. Use type="float" for float process variables. *Exponent:* Exponent of base 10 to specify fix point precision of simpleType = Uint8, Uint16, Uint32, Int8, Int16, Int32 (see chapter 6.5.1). The stack value will be multiplied by 10 Exp to get the value of the process variable. value of the process variable = 10^{Exp} * stack value The exponent therefore specifies the position of comma within a fix point value. Following values are possible: *FormatString:* String that defines the value output format. For a list of available format options, see chapter 6.5. *Value:* Default value of a process variable. Only available for independent process variables (no type specified).

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

The decimal places of the default value are defining the exponent.
If no decimal places but a fixed point format is given, the exponent is defined by the decimal places of the fixed point format. Therefore we recommend to always specify an exponent if you use a fixed point format with decimal places.

Type:
Defines the calculation method:

Value	Calculated during EventStates processing, "Get" command or "LD" in another Process Variable. Pay attention on the command order inside an instruction list!
Process	Calculated on EventHandler Process (see chapter 3.7.1)

Precision:
Defines the precision of the RPN instructions (option).
If no precision is specified, the stack result will have the same precision as the first loaded value (exp/precision of source).

Following values are possible:

precision 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

RPN Instruction List
The 'RPN Instruction List' defines how the value of the process variable is being calculated.

Example

Process variable that negates the status of input P0:

```
<Alarm_0_PV>
   <Value>
      <LDN _="/Process/MB/IO/I/P0"/>
   </Value>
</Alarm_0_PV >
```

Process Variable loading an analog value to reformat the displayed value.

```
<Alarm_1_PV format="F2,1;V">
   <Value>
      <LDN _="/Process/MB/A/AI/P0"/>
   </Value>
</Alarm_1_PV>
```

Process Variable loading an analog value with exponent.

```
<Alarm_2_PV exp="-2">
   <Value>
      <LDN _="/Process/MB/A/AI/P0"/>
   </Value>
</Alarm_2_PV>
```

Independent process variable with default value.

```
<VariableInit def="250">
```

Example with float process variables:
Task: Compute the pressure of a pressure sensor from a 0..20mA analog input.
The maximum of the pressure sensor is 6000 mbar.
Analog input is from S1-AE3 module, channel 1 at bus address 392.
The analog input must be configured using the Periphery database:
Numerator = 2000 (end value from analog input @ 20 mA)
Denominator = 2047 (maximum raw value of A/D converter (11 bit)
Computaton of scaling factor:
D_max = 6000 (maximum pressure @ 20 mA)
AImin = 400 (start value from analog input @ 4 mA)
AImax = 2000 (end value from analog input @ 20 mA
AIcorr_max = AImax – AImin = 2000 - 400 = 1600
Factor = Dmax / AIkorr_max = 6000 / 1600 = 3.75

```
<Pressure_1 type="float" format="F.1" >
  <Value>
     <!-- Load input value -->

     <!-- S1-AE3 module at address 392 -->
     <LD _="/Process/C392/AI_AAA/P0" />

     <!-- save the value on the float stack -->
     <I2F/>

     <!-- subtract start value (4..20mA sensor) -->
     <SUBF _="400" />

     <!-- multiply by factor to get real pressure -->
     <MULF _="3.75" />

  </Value>
</Pressure_1>
```

6.3.1 RPN Instruction List

Reverse Polish notation (RPN) is a mathematical notation wherein every operator follows all of its operands, in contrast to Polish notation, which puts the operator in the prefix position. RPN is also known as Postfix notation and is parenthesis-free as long as operator arities are fixed. The description "Polish" refers to the nationality of logician JAN ŁUKASIEWICZ, who invented the Polish notation in the 1920s.

The RPN Instruction List is a FORTH (http://www.forth.org/) program that calculates the value of a process variable. Its notation is similar to the Instruction List language used by PLCs. The calculation of the value therefore works like that in PLCs using a simple calculation stack.

Database path: /PROCCFG/ProcessVars

Instruction List

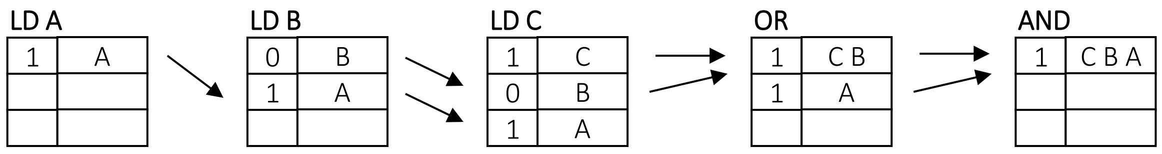
```
<ProcessVars>
   <Alarm_0_PV>
      <Value>
         <LD _="/Process/MB/IO/I/P0"/>
         <LD _="/Process/MB/IO/I/P1"/>
         <LD _="/Process/MB/IO/I/P2"/>
         <OR/>
         <AND/>
      </Value>
   </Alarm_0_PV>
</ProcessVars>
```

The instructions are adding, replacing or moving items of the stack.

The following example shows the stack operation:

LD	A	; Load Bit A
LD	B	; Load Bit B
LD	C	; Load Bit C
OR		; C or B
AND		; (C or B) and A

This program reads the three bit variables A, B and C and puts these onto the stack. The OR operation combines the items C and B and puts the result back onto the stack. Then the residual upper two stack items (result of B+C as well as A) are combined by an AND operation. The result is 1.



The stack has a maximum size of 10 items. If an error occurs (stackoverflow, stackunderflow) the result value is 0.
If values remain on the stack, only the top value will be used for further processing or during a "Get" command.

6.3.1.1 Logical instructions

INPUT	A	0	0	1	1	1234	4321	0
	B	*0*	*1*	*0*	*1*	*4321*	*0*	*1234*
OUTPUT	LD A	0	0	1	1	1234	4321	0
	NOT A	*1*	*1*	*0*	*0*	*0*	*0*	*1*
	LDN A	*1*	*1*	*0*	*0*	*0*	*0*	*1*
	DLDN A	*-1*	*-1*	*-2*	*-2*	*-1235*	*-4322*	*-1*
	*A **AND** B*	*0*	*0*	*0*	*1*	*1*	*0*	*0*
	*A **DAND** B*	*0*	*0*	*0*	*1*	*192*	*0*	*0*
	*A **ANDN** B*	*0*	*0*	*1*	*0*	*0*	*1*	*0*
	A DANDN B	*0*	*0*	*1*	*0*	*1042*	*4321*	*0*
	*A **OR** B*	*0*	*1*	*1*	*1*	*1*	*1*	*1*
	A DOR B	*0*	*1*	*1*	*1*	*5363*	*4321*	*1234*
	A ORN B	*1*	*0*	*1*	*1*	*1*	*1*	*0*
	*A **DORN** B*	*-1*	*-2*	*-1*	*-1*	*-4130*	*-1*	*-1235*
	*A **XOR** B*	*0*	*1*	*1*	*0*	*0*	*1*	*1*
	*A **DXOR** B*	*0*	*1*	*1*	*0*	*5171*	*4321*	*1234*
	A XORN B	*1*	*0*	*0*	*1*	*1*	*0*	*0*
	A DXORN B	*-1*	*-2*	*-2*	*-1*	*-5172*	*-4322*	*-1235*

LD/LDF *- Load value (DWORD / float)*	
Syntax	**`<LD _="`*`SystemProperty`*`" exp="`*`Exponent`*`"/>`**
Description	Loads the value defined by the system property path to the top of the processing stack. If the value type is "float", only the fixed part is loaded.
Elements	*SystemProperty:* Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14). May also be a constant. *Exponent:* See chapter 6.3. An exponent within LD instruction may be necessary for constants used for operations with variables which already have an exponent (e.g. PLC variables).
Example	Load the bit value "1" of the first digital input of a FP Gateway. `<LD _="/Process/MB/IO/I/P0"/>` Stack result: 1

Load the word value 1234 of a PLC variable.

```
<LD _="/Process/Bus1/Device_0/Variable_0"/>
```

Stack result: 1234

Load several PLC variables with value Variable_0=10, Variable_1=0, Variable_2=3.

```
<LD v1="/Process/Bus1/Device_0/Variable_0"
  v2="/Process/Bus1/Device_0/Variable_1"
  v3="/Process/Bus1/Device_0/Variable_2"/>
```

Stack items:

3
0
10

Stack result: 3

Load constant "13" to the top of the stack.

```
<LD _="13"/>
```

Load constant "0.013" to the top of the stack.

```
<LD _="13" exp="-3"/>
```

***LDN/NOT** - Load value and logically negate (DWORD only)*	

Syntax	**`<LDN _="SystemProperty"/>`**
	or
	`<NOT _="SystemProperty"/>`

Description	Reads the value defined by the address and loads its logical negation at the top of the processing stack.
Elements	*SystemProperty:* Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14). May also be a constant.

Example	Reads the bit value "0" of the first FP Gateway digital input and loads its logical negation on the top of the stack.
	`<LDN _="/Process/C40/MB/IO/I/P0"/>`
	Stack result: 1
	
	Reads the word value "1234" of a PLC variable and loads its logical negation on the top of the stack.
	`<LDN _="/Process/Bus1/Device_0/Variable_0"/>`
	Stack result: 0

***DLDN/NEG** - Load value and binary negate (DWORD only)*	

Syntax	**`<DLDN _="SystemProperty"/>`**
	or
	`<NEG _="SystemProperty"/>`

Description	Reads the value defined by the address and loads the result of a 32bit binary negation at the top of the processing stack.
Elements	*SystemProperty:* Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14). May also be a constant.

Example	Reads the word value "1234" of a PLC variable and loads the result of its 32bit binary negation on the top of the stack. `<DLDN _="/Process/Bus1/Device_0/Variable_0"/>` "1234" as a 32bit binary value: `00000000000000000000010011010010` Result of binary negation: `11111111111111111111101100101101` Stack result: -1235

LDS - Load Special (DWORD only)	
Syntax	**`<LDS _="PLCVariable" AddInfo="ErrorCode"/>`**
Description	Loads the additional info of a PLC variable at the top of the processing stack. The result is a 32bit value calculated by the ErrorClass and ErrorValue of the variable. See PLC-TiXML-Manual for further information.
Elements	*PLCVariable:* Path to a PLC variable on the `/Process/BusX/` system property branch. Not all PLCs do support additional error codes. *ErrorCode:* see PLC-TiXML- Manual

Example	Load the ErrorClass and ErrorNumber information of the PLC variable "Variable_0" of "Device_0" on PLC-Bus "Bus1" on the top of the stack.
	`<LDS _="/Process/Bus1/Device_0/Variable_0"`
	`AddInfo="Error"/>`

AND - Load value and logically AND with value (DWORD only)	
Syntax	**`<AND v1="SystemProperty" v2="SystemProperty"/>`**
Description	Combination of the instructions LD and AND. It reads the values defined by both addresses. Thereafter a logical AND operation between these two items is done and the result of the operation is written on the top of the stack.
Elements	*SystemProperty:* Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14). May also be a constant.

Example	Reads the bit value of the first two FP Gateway digital inputs and loads the result of the logically AND operation on the top of the stack.
	`<AND v1="/Process/MB/IO/I/P0"`
	`v2="/Process/MB/IO/I/P0"/>`
	or
	`<LD _="/Process/MB/IO/I/P0"/>`
	`<AND _="/Process/MB/IO/I/P1"/>`
	or
	`<LD _="/Process/MB/IO/I/P0"/>`
	`<LD _="/Process/MB/IO/I/P1"/>`
	`<AND/>`
	
	See table on the top of this chapter for logic operation results.

DAND - Load value and binary AND with value (DWORD only)

Syntax	**`<DAND v1="SystemProperty" v2="SystemProperty"/>`**
Description	Combination of the instructions LD and DAND. It reads the values defined by both addresses. Thereafter a binary AND operation between these two items is done and the result of the operation is written on the top of the stack.
Elements	*SystemProperty:* Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14). May also be a constant.
Example	Reads the word values Variable_0=1234 and Variable_1=4321 of a PLC and loads the result of the binary AND operation on the top of the stack.

```
<DAND v1="/Process/Bus1/Device_0/Variable_0"
   v2="/Process/Bus1/Device_0/Variable_1"/>
```
or
```
<LD _="/Process/Bus1/Device_0/Variable_0"/>
<DAND _="/Process/Bus1/Device_0/Variable_1"/>
```
or
```
<LD _="/Process/Bus1/Device_0/Variable_0"/>
<LD _="/Process/Bus1/Device_0/Variable_1"/>
<DAND/>
```

"1234" as a 32bit binary value:	00000000000000000000010011010010
"4321" as a 32bit binary value:	00000000000000000001000011100001
Result of binary AND:	00000000000000000000000011000000

Stack result: 192

ANDN - Load value and logically AND with negated value (DWORD only)

Syntax	**`<ANDN v1="SystemProperty" v2="SystemProperty"/>`**
Description	Combination of the instructions LD, LDN and AND. It reads the first addressed value by LD and the second is read invertedly by LDN. Thereafter, a logical AND operation between these two items is done and the result of the operation is written on the top of the stack.
Elements	*SystemProperty:* Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14). May also be a constant.
Example	The result of the logically AND operation is loaded on the top of the stack.

```
<ANDN v1="/Process/MB/IO/I/P0"
v2="/Process/MB/IO/I/P0"/>
```
or
```
<LD _="/Process/MB/IO/I/P0"/>
<ANDN _="/Process/MB/IO/I/P1"/>
```
or
```
<LD _="/Process/MB/IO/I/P0"/>
<LDN _="/Process/MB/IO/I/P1"/>
<AND/>
```
See table on the top of this chapter for logic operation results.
Reads the bit value of the first two FP Gateway digital inputs, whereby the second becomes inverted.

***DANDN** - Load value and binary AND with negated value (DWORD only)*	
Syntax	**`<DANDN v1="SystemProperty" v2="SystemProperty"/>`**
Description	Combination of the instructions LD, DLDN and DAND. It reads the first addressed value by LD and the second is read binary invertedly by DLDN. Thereafter, a binary AND operation between these two items is done and the result of the operation is written on the top of the stack.
Elements	*SystemProperty:* Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14). May also be a constant.

Example	Reads the word values Variable_0=1234 and Variable_1=4321 of a PLC and loads the result of the binary AND operation on the top of the stack.
	`<DANDN v1="/Process/Bus1/Device_0/Variable_0"`
	`v2="/Process/Bus1/Device_0/Variable_1"/>`
	or
	`<LD _="/Process/Bus1/Device_0/Variable_0"/>`
	`<DANDN _="/Process/Bus1/Device_0/Variable_1"/>`
	or
	`<LD _="/Process/Bus1/Device_0/Variable_0"/>`
	`<DLDN _="/Process/Bus1/Device_0/Variable_1"/>`
	`<DAND/>`
	
	"1234" as a 32bit binary value: `00000000000000000000010011010010`
	32bit binary negated value of "4321":`11111111111111111110111100011110`
	Result of binary AND:
	`00000000000000000000010000010010`
	Stack result: 1042

***OR** - Load value and logically OR with value (DWORD only)*	
Syntax	**`<OR v1="SystemProperty" v2="SystemProperty"/>`**
Description	Combination of the instructions LD and OR. It reads the values defined by both addresses. Thereafter a logical OR operation between these two items is done and the result of the operation is written on the top of the stack.
Elements	*SystemProperty:* Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14). May also be a constant.

Example	Reads the bit value of the first two FP Gateway digital inputs and loads the result of the logically OR operation on the top of the stack.
	`<OR v1="/Process/MB/IO/I/P0"`
	`v2="/Process/MB/IO/I/P0"/>`
	or
	`<LD _="/Process/MB/IO/I/P0"/>`
	`<OR _="/Process/MB/IO/I/P1"/>`
	or
	
	`<LD _="/Process/MB/IO/I/P0"/>`
	`<LD _="/Process/MB/IO/I/P1"/>`
	`<OR/>`
	
	See table on the top of this chapter for logic operation results.

***DOR** - Load value and binary OR with value (DWORD only)*

Syntax

```
<DOR v1="SystemProperty" v2="SystemProperty"/>
```

Description

Combination of the instructions LD and DOR. It reads the values defined by both addresses. Thereafter a binary OR operation between these two items is done and the result of the operation is written on the top of the stack.

Elements

SystemProperty:
Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14). May also be a constant.

Example

Reads the word values Variable_0=1234 and Variable_1=4321 of a PLC and loads the result of the binary OR operation on the top of the stack.

```
<DOR v1="/Process/Bus1/Device_0/Variable_0"
v2="/Process/Bus1/Device_0/Variable_1"/>
```

or

```
<LD _="/Process/Bus1/Device_0/Variable_0"/>
<DOR _="/Process/Bus1/Device_0/Variable_1"/>
```

or

```
<LD _="/Process/Bus1/Device_0/Variable_0"/>
<LD _="/Process/Bus1/Device_0/Variable_1"/>
<DOR/>
```

"1234" as a 32bit binary value: 00000000000000000000010011010010
"4321" as a 32bit binary value: 00000000000000000001000011100001
Result of binary OR: 00000000000000000001010011110011
Stack result: 5363

***ORN** - Load value and logically OR with negated value (DWORD only)*

Syntax

```
<ORN v1="SystemProperty" v2="SystemProperty"/>
```

Description

Combination of the instructions LD, LDN and OR. It reads the first addressed value by LD and the second is read inverted by LDN. Thereafter a logical OR operation between these two items is done and the result of the operation is written on the top of the stack.

Elements

SystemProperty:
Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14). May also be a constant.

Example

Reads the bit value of the first two FP Gateway digital inputs whereby the second becomes inverted. The result of the logically OR operation is loaded on the top of the stack.

```
<ORN v1="/Process/MB/IO/I/P0"
v2="/Process/MB/IO/I/P0"/>
```

or

```
<LD _="/Process/MB/IO/I/P0"/>
<ORN _="/Process/MB/IO/I/P1"/>
```

or

```
<LD _="/Process/MB/IO/I/P0"/>
<LDN _="/Process/MB/IO/I/P1"/>
<OR/>
```

See table on the top of this chapter for logic operation results.

DORN *- Load value and binary OR with negated value (DWORD only)*	
Syntax	**`<DORN v1="SystemProperty" v2="SystemProperty"/>`**
Description	Combination of the instructions LD, DLDN and DOR. It reads the first addressed value by LD and the second is read binary inverted by DLDN. Thereafter a binary OR operation between these two items is done and the result of the operation is written on the top of the stack.
Elements	*SystemProperty:* Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14). May also be a constant.

Example	Reads the word values Variable_0=1234 and Variable_1=4321 of a PLC and loads the result of the binary OR operation on the top of the stack.
	`<DORN v1="/Process/Bus1/Device_0/Variable_0"`
	`v2="/Process/Bus1/Device_0/Variable_1"/>`
	or
	`<LD _="/Process/Bus1/Device_0/Variable_0"/>`
	`<DORN _="/Process/Bus1/Device_0/Variable_1"/>`
	or
	`<LD _="/Process/Bus1/Device_0/Variable_0"/>`
	`<DLDN _="/Process/Bus1/Device_0/Variable_1"/>`
	`<DOR/>`
	
	"1234" as a 32bit binary value: `00000000000000000000010011010010`
	32bit binary negated value of "4321":`11111111111111111110111100011110`
	Result of binary OR: `11111111111111111110111111011110`
	Stack result: -4130

XOR *- Load value and logically XOR with value (DWORD only)*	
Syntax	**`<OR v1="SystemProperty" v2="SystemProperty"/>`**
Description	Combination of the instructions LD and XOR. It reads the values defined by both addresses. Thereafter a logical XOR operation between these two items is done and the result of the operation is written on the top of the stack.
Elements	*SystemProperty:* Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14). May also be a constant.

Example	Reads the bit value of the first two FP Gateway digital inputs and loads the result of the logically XOR operation on the top of the stack.
	`<XOR v1="/Process/MB/IO/I/P0"`
	`v2="/Process/MB/IO/I/P0"/>`
	or
	`<LD _="/Process/MB/IO/I/P0"/>`
	`<XOR _="/Process/MB/IO/I/P1"/>`
	or
	
	`<LD _="/Process/MB/IO/I/P0"/>`
	`<LD _="/Process/MB/IO/I/P1"/>`
	`<XOR/>`
	
	See table on the top of this chapter for logic operation results.

***DXOR** - Load value and binary XOR with value (DWORD only)*

Syntax

```
<DOR v1="SystemProperty" v2="SystemProperty"/>
```

Description: Combination of the instructions LD and DXOR. It reads the values defined by both addresses. Thereafter a binary XOR operation between these two items is done and the result of the operation is written on the top of the stack.

Elements

SystemProperty
Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14). May also be a constant.

Example

Reads the word values Variable_0=1234 and Variable_1=4321 of a PLC and loads the result of the binary XOR operation on the top of the stack.

```
<DXOR v1="/Process/Bus1/Device_0/Variable_0"
   v2="/Process/Bus1/Device_0/Variable_1"/>
```

or

```
<LD _="/Process/Bus1/Device_0/Variable_0"/>
<DXOR _="/Process/Bus1/Device_0/Variable_1"/>
```

or

```
<LD _="/Process/Bus1/Device_0/Variable_0"/>
<LD _="/Process/Bus1/Device_0/Variable_1"/>
<DXOR/>
```

"1234" as a 32bit binary value:	00000000000000000000010011010010
"4321" as a 32bit binary value:	00000000000000000001000011100001
Result of binary XOR:	00000000000000000001010011110011

Stack result: 5363

***XORN** - Load value and logically XOR with negated value (DWORD only)*

Syntax

```
<XORN v1="SystemProperty" v2="SystemProperty"/>
```

Description Combination of the instructions LD, LDN and XOR. It reads the first addressed value by LD and the second is read inverted by LDN. Thereafter a logical XOR operation between these two items is done and the result of the operation is written on the top of the stack.

Elements

SystemProperty:
Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14). May also be a constant.

Example

Reads the bit value of the first two FP Gateway digital inputs whereby the second becomes inverted. The result of the logically XOR operation is loaded on the top of the stack.

```
<XORN v1="/Process/MB/IO/I/P0"
v2="/Process/MB/IO/I/P0"/>
```

or

```
<LD _="/Process/MB/IO/I/P0"/>
<XORN _="/Process/MB/IO/I/P1"/>
```

or

```
<LD _="/Process/MB/IO/I/P0"/>
<LDN _="/Process/MB/IO/I/P1"/>
<XOR/>
```

See table on the top of this chapter for logic operation results.

***DXORN** - Load value and binary XOR with negated value (DWORD only)*

Syntax **`<DANDN v1="SystemProperty" v2="SystemProperty"/>`**

Description Combination of the instructions LD, DLDN and DXOR. It reads the first addressed value by LD and the second is read binary inverted by DLDN. Thereafter a binary XOR operation between these two items is done and the result of the operation is written on the top of the stack.

Elements *SystemProperty:*
Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14). May also be a constant.

Example Reads the word values Variable_0=1234 and Variable_1=4321 of a PLC and loads the result of the binary XOR operation on the top of the stack.

```
    <DXORN v1="/Process/Bus1/Device_0/Variable_0"
        v2="/Process/Bus1/Device_0/Variable_1"/>
```
or
```
    <LD _="/Process/Bus1/Device_0/Variable_0"/>
    <DXORN _="/Process/Bus1/Device_0/Variable_1"/>
```
or
```
    <LD _="/Process/Bus1/Device_0/Variable_0"/>
    <DLDN _="/Process/Bus1/Device_0/Variable_1"/>
    <DXOR/>
```

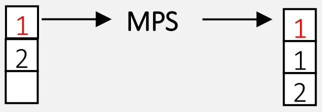
"1234" as a 32bit binary value: 00000000000000000000010011010010
32bit binary negated value of "4321":11111111111111111110111100011110
Result of binary XOR: 11111111111111111110101111001100
Stack result: -5172

6.3.1.2 Stack operations

***MPS/MPSF** – Multiply stack (DWORD) / (float)*

Syntax **`<MPS/>`**

Description Multiplies a stack value to use it for two operations.

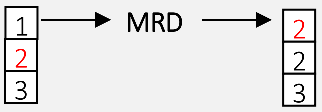


Example Reads the bit value of the first FP Gateway digital input, multiply it and set two output ports.
```
<LD _="/Process/MB/IO/I/P0"/>
<MPS/>
<ST _="/Process/MB/IO/Q/P0"/>
<ST _="/Process/MB/IO/Q/P1"/>
```

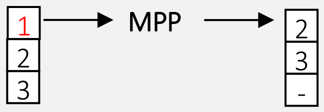
***MRD** – Copy 2nd stack level to top of stack (DWORD only)*

Syntax **`<MRD/>`**

Description Replaces the value on the top of the stack with the value from the second stack level.



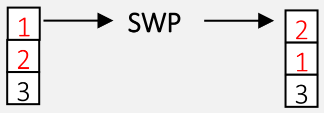
Example	Load value 2 and 5 on the stack, MRD the second stack level (value 2) over the first stack leven (value 5). The result of the ADD operation is 4 (2+2) because value 5 was replaced by value 2.
	`<LD _="2"/>`
	`<LD _="5"/>`
	`<MRD/>`
	`<ADD/>`

MPP/MPPF – Remove value at the top of stack (DWORD) / (float)	
Syntax	**`<MPP/>`**
Description	Removes the value on the top of the stack. 

Example	Reads the bit values of the first two FP Gateway digital inputs. MPP removes the top stack level value (which is input P1). Therefore, the value of input P0 will be written to output P0.
	`<LD _="/Process/MB/IO/I/P0"/>`
	`<LD _="/Process/MB/IO/I/P1"/>`
	`<MPP/>`
	`<ST _="/Process/MB/IO/Q/P0"/>`

CPY – Copy value (DWORD only)	
Syntax	**`<CPY _="`*`SystemProperty`*`"/>`**
Description	Copies the value at the top of the processing stack to the given system property address. The stack remains unchanged.
Elements	*SystemProperty:* Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14).

Example	Copy the value of input P0 to output P0 and P1.
	`<LD _="/Process/MB/IO/I/P0"/>`
	`<CPY _="/Process/MB/IO/Q/P0"/>`
	`<CPY _="/Process/MB/IO/Q/P1"/>`

SWP/SWPF – swap values (DWORD) / (float)	
Syntax	***`<SWP/>`***
Description	It swaps the two values at the top of the processing stack. 

Example	Reads the bit values of the first two FP Gateway digital inputs. SWP swaps the upper two stack items. Therefore, the value of input P0 will be written to output P0.
	`<LD _="/Process/MB/IO/I/P0"/>`
	`<LD _="/Process/MB/IO/I/P1"/>`
	`<SWP/>`
	`<ST _="/Process/MB/IO/Q/P0"/>`

ST/STF – Store (DWORD) / (float)	
Syntax	**`<ST _="SystemProperty"/>`**
Description	Stores the value from the top of the processing stack to the given system property address. The stored value becomes removed from the stack.
Elements	*SystemProperty:* Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14).

Example	Store the value of input P0 to output P0.
	`<LD _="/Process/MB/IO/I/P0"/>`
	`<ST _="/Process/MB/IO/Q/P0"/>`

I2F – Copies value from top of DWORD stack to top of FLOAT stack	
Syntax	**`<I2F/>`**
Description	Copies the value from the top of the DWORD stack, converts the value to float and copies it to the top of the FLOAT stack.

F2I – Copies value from top of FLOAT stack to top of DWORD stack	
Syntax	**`<F2I/>`**
Description	Copies the value from the top of the FLOAT stack, converts the value to DWORD and copies it to the top of the DWORD stack. Please note that information may get lost ! Example: Top of DWORD stack = 104 Top of FLOAT stack = 221.25 After issuing the command <F2I/> the top of the DWORD stack contains 221. The fraction (.25) gets lost in this case !

6.3.1.3 Comparison instructions

INPUT	A	0	0	1	-1	1
	B	*0*	*1*	*0*	*1*	*-1*
OUTPUT	A **GT** B	0	0	1	1	0
	*A **GTI** B*	*0*	*0*	*1*	*0*	*1*
	*A **LT** B*	*0*	*1*	*0*	*0*	*1*
	*A **LTI** B*	*0*	*1*	*0*	*1*	*0*
	*A **EQ** B*	*1*	*0*	*0*	*0*	*0*
	*A **NE** B*	*0*	*1*	*1*	*1*	*1*
	*A **GE** B*	*1*	*0*	*1*	*1*	*0*
	*A **GEI** B*	*1*	*0*	*1*	*0*	*1*
	*A **LE** B*	*1*	*1*	*0*	*0*	*1*
	*A **LEI** B*	*1*	*1*	*0*	*1*	*0*
	A MIN B	*0*	*0*	*0*	*1*	*1*
	*A **MINI** B*	*0*	*0*	*0*	*-1*	*-1*
	A MAX B	*0*	*1*	*1*	*-1*	*-1*
	*A **MAXI** B*	*0*	*1*	*1*	*1*	*1*

***GT/GTI** – greater than (unsigned values) / (signed values) (DWORD only)*

Syntax

```
<GT v1="SystemProperty" v2="SystemProperty"/>
<GTI v1="SystemProperty" v2="SystemProperty"/>
```

Description

Compares both values. If v1 is greater than v2, a 1 will be written to the top of the processing stack.

Elements

SystemProperty:
Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14). May also be a constant.

Example

Compares the unsigned word value Variable_0=1234 of a PLC with a constant value and loads the logical result (0/1) of the comparison on the top of the stack.

```
<GT v1="/Process/Bus1/Device_0/Variable_0" v2="100"/>
```

or

```
    <LD _="/Process/Bus1/Device_0/Variable_0"/>
    <GT _="100"/>
```

or

```
    <LD _="/Process/Bus1/Device_0/Variable_0"/>
    <LD _="100"/>
    <GT/>
```

Result: 1 (TRUE)

Compares the signed word value Variable_0=-1234 of a PLC with a constant value and loads the logical result (0/1) of the comparison on the top of the stack.

```
<GTI v1="/Process/Bus1/Device_0/Variable_0" v2="-100"/>
```

	Result: 0 (FALSE) Compares the unsigned word values Variable_0=1234 and Variable_1=4321 of a PLC and loads the logical result (0/1) of the comparison on the top of the stack. `<GT v1="/Process/Bus1/Device_0/Variable_0" v2="/Process/Bus1/Device_0/Variable_1"/>` Result: 0 (FALSE)

***LT/LTI** – less than (unsigned values) / (signed values) (DWORD only)*	

Syntax	***`<LT v1="SystemProperty" v2="SystemProperty"/>`***
	`<LTI v1="SystemProperty" v2="SystemProperty"/>`

Description	Compares both values. If v1 is less than v2, a 1 will be written to the top of the processing stack.
Elements	*SystemProperty:* Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14). May also be a constant.

Example	Compares the unsigned word value Variable_0=1234 of a PLC with a constant value and loads the logical result (0/1) of the comparison on the top of the stack.
	`<LT v1="/Process/Bus1/Device_0/Variable_0" v2="100"/>`
	or
	    `<LD _="/Process/Bus1/Device_0/Variable_0"/>`
	    `<LT _="100"/>`
	or
	    `<LD _="/Process/Bus1/Device_0/Variable_0"/>`
	    `<LD _="100"/>`
	    `<LT/>`
	Result: 0 (FALSE)
	
	Compares the signed word value Variable_0=-1234 of a PLC with a constant value and loads the logical result (0/1) of the comparison on the top of the stack.
	`<LTI v1="/Process/Bus1/Device_0/Variable_0" v2="-100"/>`
	Result: 1 (TRUE)
	
	Compares the unsigned word values Variable_0=1234 and Variable_1=4321 of a PLC and loads the logical result (0/1) of the comparison on the top of the stack.
	`<LT v1="/Process/Bus1/Device_0/Variable_0" v2="/Process/Bus1/Device_0/Variable_1"/>`
	Result: 1 (TRUE)

***EQ** – equal (DWORD only)*	
Syntax	***`<EQ v1="SystemProperty" v2="SystemProperty"/>`***
Description	Compares both values. If v1 is equal v2, a 1 will be written to the top of the processing stack.
Elements	*SystemProperty:* Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14). May also be a constant.
Example	Compares the unsigned word value Variable_0=1234 of a PLC with a constant value and loads the logical result (0/1) of the comparison on the top of the stack.

	`<EQ v1="/Process/Bus1/Device_0/Variable_0" v2="100"/>`
	or
	`    <LD _="/Process/Bus1/Device_0/Variable_0"/>`
	`    <EQ _="100"/>`
	or
	`    <LD _="/Process/Bus1/Device_0/Variable_0"/>`
	`    <LD _="100"/>`
	`    <EQ/>`
	Result: 0 (FALSE)
	
	Compares the signed word value Variable_0=-1234 of a PLC with a constant value and loads the logical result (0/1) of the comparison on the top of the stack.
	`<EQ v1="/Process/Bus1/Device_0/Variable_0" v2="-1234"/>`
	Result: 1 (TRUE)
	
	Compares the unsigned word values Variable_0=1234 and Variable_1=4321 of a PLC and loads the logical result (0/1) of the comparison on the top of the stack.
	`<EQ v1="/Process/Bus1/Device_0/Variable_0" v2="/Process/Bus1/Device_0/Variable_1"/>`
	Result: 0 (FALSE)

NE – not equal (DWORD only)	
Syntax	***`<NE v1="SystemProperty" v2="SystemProperty"/>`***
Description	Compares both values. If v1 is not equal v2, a 1 will be written to the top of the processing stack.
Elements	*SystemProperty:* Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14). May also be a constant.

Example	Compares the unsigned word value Variable_0=1234 of a PLC with a constant value and loads the logical result (0/1) of the comparison on the top of the stack.
	`<NE v1="/Process/Bus1/Device_0/Variable_0" v2="100"/>`
	or
	`    <LD _="/Process/Bus1/Device_0/Variable_0"/>`
	`    <NE _="100"/>`
	or
	`    <LD _="/Process/Bus1/Device_0/Variable_0"/>`
	`    <LD _="100"/>`
	`    <NE/>`
	Result: 1 (TRUE)
	
	Compares the signed word value Variable_0=-1234 of a PLC with a constant value and loads the logical result (0/1) of the comparison on the top of the stack.
	`<NE v1="/Process/Bus1/Device_0/Variable_0" v2="-1234"/>`
	Result: 0 (FALSE)
	
	Compares the unsigned word values Variable_0=1234 and Variable_1=4321 of a PLC and loads the logical result (0/1) of the comparison on the top of the stack.
	`<NE v1="/Process/Bus1/Device_0/Variable_0" v2="/Process/Bus1/Device_0/Variable_1"/>`

Result: 1 (TRUE)

GE/GEI – greater equal (unsigned values) / (signed values) (DWORD only)

Syntax

```
<GE v1="SystemProperty" v2="SystemProperty"/>
<GEI v1="SystemProperty" v2="SystemProperty"/>
```

Description

Compares both values. If v1 is greater than or equal v2, a 1 will be written to the top of the processing stack.

Elements

SystemProperty:
Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14). May also be a constant.

Example

Compares the unsigned word value Variable_0=1234 of a PLC with a constant value and loads the logical result (0/1) of the comparison on the top of the stack.

```
<GE v1="/Process/Bus1/Device_0/Variable_0" v2="100"/>
```

or

```
    <LD _="/Process/Bus1/Device_0/Variable_0"/>
    <GE _="100"/>
```

or

```
    <LD _="/Process/Bus1/Device_0/Variable_0"/>
    <LD _="100"/>
    <GE/>
```

Result: 1 (TRUE)

Compares the signed word value Variable_0=-1234 of a PLC with a constant value and loads the logical result (0/1) of the comparison on the top of the stack.

```
<GEI v1="/Process/Bus1/Device_0/Variable_0" v2="-1234"/>
```

Result: 1 (TRUE)

Compares the unsigned word values Variable_0=1234 and Variable_1=4321 of a PLC and loads the logical result (0/1) of the comparison on the top of the stack.

```
<GE v1="/Process/Bus1/Device_0/Variable_0"
v2="/Process/Bus1/Device_0/Variable_1"/>
```

Result: 0 (FALSE)

LE/LEI – less equal (unsigned values) / (signed values) (DWORD only)

Syntax

```
<LE v1="SystemProperty" v2="SystemProperty"/>
<LEI v1="SystemProperty" v2="SystemProperty"/>
```

Description

Compares both values. If v1 is less than or equal v2, a 1 will be written to the top of the processing stack.

Elements

SystemProperty:
Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14). May also be a constant.

Example

Compares the unsigned word value Variable_0=1234 of a PLC with a constant value and loads the logical result (0/1) of the comparison on the top of the stack.

```
<LE v1="/Process/Bus1/Device_0/Variable_0" v2="100"/>
```

or

```
    <LD _="/Process/Bus1/Device_0/Variable_0"/>
    <LE _="100"/>
```

or

	`<LD _="/Process/Bus1/Device_0/Variable_0"/>`
	`<LD _="100"/>`
	`<LE/>`
	Result: 0 (FALSE)
	
	Compares the signed word value Variable_0=-1234 of a PLC with a constant value and loads the logical result (0/1) of the comparison on the top of the stack.
	`<LEI v1="/Process/Bus1/Device_0/Variable_0" v2="-1234"/>`
	Result: 1 (TRUE)
	
	Compares the unsigned word values Variable_0=1234 and Variable_1=4321 of a PLC and loads the logical result (0/1) of the comparison on the top of the stack.
	`<LE v1="/Process/Bus1/Device_0/Variable_0" v2="/Process/Bus1/Device_0/Variable_1"/>`
	Result: 1 (TRUE)

	MIN/MINI – minimum (unsigned values) / signed (DWORD only)

Syntax	***`<MIN v1="SystemProperty" v2="SystemProperty"/>`***
	`<MINI v1="SystemProperty" v2="SystemProperty"/>`

Description	Compares both values and removes the greater one from the stack.
Elements	*SystemProperty:* Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14). May also be a constant.

Example	Compares the unsigned word values Variable_0=1234 and Variable_1=4321 of a PLC and loads the lesser value on the top of the stack.
	`<MIN v1="/Process/Bus1/Device_0/Variable_0"`
	
	`v2="/Process/Bus1/Device_0/Variable_1"/>`
	or
	`<LD _="/Process/Bus1/Device_0/Variable_0"/>`
	`<MIN _="/Process/Bus1/Device_0/Variable_1"/>`
	or
	`<LD _="/Process/Bus1/Device_0/Variable_0"/>`
	`<LD _="/Process/Bus1/Device_0/Variable_1"/>`
	`<MIN/>`
	Result: 1234
	
	Compares the signed word values Variable_0=1234 and Variable_2=-4321 of a PLC and loads the lesser value on the top of the stack.
	`<MINI v1="/Process/Bus1/Device_0/Variable_0"`
	
	`v2="/Process/Bus1/Device_0/Variable_2"/>`
	Result: -4321

	MAX/MAXI – maximum (unsigned values) / (signed values) (DWORD only)

Syntax	***`<MAX v1="SystemProperty" v2="SystemProperty"/>`***
	`<MAXI v1="SystemProperty" v2="SystemProperty"/>`

Description	Compares both values and removes the lesser one from the stack.
Elements	*SystemProperty:*

Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14). May also be a constant.

Example

Compares the unsigned word values Variable_0=1234 and Variable_1=4321 of a PLC and loads the greater value on the top of the stack.

```
    <MAX v1="/Process/Bus1/Device_0/Variable_0"

    v2="/Process/Bus1/Device_0/Variable_1"/>
```

or

```
    <LD _="/Process/Bus1/Device_0/Variable_0"/>
    <MAX _="/Process/Bus1/Device_0/Variable_1"/>
```

or

```
    <LD _="/Process/Bus1/Device_0/Variable_0"/>
    <LD _="/Process/Bus1/Device_0/Variable_1"/>
    <MAX/>
```

Result: 4321

Compares the signed word values Variable_0=1234 and Variable_2=-4321 of a PLC and loads the lesser value on the top of the stack.

```
    <MAXI v1="/Process/Bus1/Device_0/Variable_0"

    v2="/Process/Bus1/Device_0/Variable_2"/>
```

Result: 1234

6.3.1.4 Math operations

If the input value is a signed value or the math operation may result in a signed value, a signed value operation (ADDI/SUBI/MULI/DIVI) must be used.

INPUT	A	1234	4321	1234	-1234	-4321
	B	4321	1234	-4321	4321	-1234
OUTPUT	**ADD** A B	5555	5555			
	ADDI A B	5555	5555	-3087	3087	-5555
	SUB A B		3087			
	SUBI A B	-3087	3087	5555	-5555	-3087
	MUL A B	5332114	5332114			
	MULI A B	5332114	5332114	-5332114	-5332114	5332114
	DIV A B	0	3			
	DIVI A B	0	3	0	0	3

ADD/ADDI/ADDF – addition (unsigned values) / (signed values) / (float)	
Syntax	***<ADD v1="SystemProperty" v2="SystemProperty"/>*** ***<ADDI v1="SystemProperty" v2="SystemProperty"/>***
Description	Adds the second stack value to the first stack value and replaces them with the result to the operation. Value = v1 + v2
Elements	*SystemProperty:* Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14). May also be a constant. If the system property is a signed value or the operation may result in a signed value, ADDI must be used.

Example	Adds the unsigned word value Variable_1=4321 to Variable_0=1234 of a PLC and loads the result on the top of the stack.
	`<ADD v1="/Process/Bus1/Device_0/Variable_0"`
	`v2="/Process/Bus1/Device_0/Variable_1"/>`
	or
	`<LD _="/Process/Bus1/Device_0/Variable_0"/>`
	`<ADD _="/Process/Bus1/Device_0/Variable_1"/>`
	or
	`<LD _="/Process/Bus1/Device_0/Variable_0"/>`
	`<LD _="/Process/Bus1/Device_0/Variable_1"/>`
	`<ADD/>`
	Result: 5555
	
	Adds the signed word value Variable_2=-1234 to Variable_1=4321 of a PLC and loads the result on the top of the stack.
	`<ADDI v1="/Process/Bus1/Device_0/Variable_1"`
	`v2="/Process/Bus1/Device_0/Variable_2"/>`
	Result: 3087

SUB/SUBI/SUBF – subtraction (unsigned values) / (signed values) / (float)	
Syntax	***<SUB v1="SystemProperty" v2="SystemProperty"/>*** ***<SUBI v1="SystemProperty" v2="SystemProperty"/>***
Description	Subtracts the second stack value from the first stack value and replaces them with the result to the operation. Value = v1 - v2
Elements	*SystemProperty:* Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14). May also be a constant. If the system property is a signed value or the operation may result in a signed value, SUBI must be used.

Example	Subtracts the unsigned word value Variable_1=1234 from Variable_0=4321 of a PLC and loads the result on the top of the stack.
	`<SUB v1="/Process/Bus1/Device_0/Variable_0"`
	
	`v2="/Process/Bus1/Device_0/Variable_1"/>`
	or

```
    <LD _="/Process/Bus1/Device_0/Variable_0"/>
    <SUB _="/Process/Bus1/Device_0/Variable_1"/>
```
or
```
    <LD _="/Process/Bus1/Device_0/Variable_0"/>
    <LD _="/Process/Bus1/Device_0/Variable_1"/>
    <SUB/>
```
Result: 3087

Subtracts the signed word value Variable_3=-1234 from Variable_2=-4321 of a PLC and loads the Result on the top of the stack.
```
    <SUBI v1="/Process/Bus1/Device_0/Variable_2"

    v2="/Process/Bus1/Device_0/Variable_3"/>
```
Result: -5555

MUL/MULI/MULF – multiplication (unsigned values) / (signed values) / (float)	
Syntax	***<MUL v1="SystemProperty" v2="SystemProperty"/>*** ***<MULI v1="SystemProperty" v2="SystemProperty"/>***
Description	Multiplies v1 by v2 and writes the result to the top of the stack. Value = v1 * v2
Elements	*SystemProperty:* Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14). May also be a constant. If the system property is a signed value or the operation may result in a signed value, MULI must be used.

Example	Multiplies the unsigned word values Variable_0=1234 and Variable_1=4321 of a PLC and loads the Result on the top of the stack.
	`<MUL v1="/Process/Bus1/Device_0/Variable_0"`
	`v2="/Process/Bus1/Device_0/Variable_1"/>`
	or
	`<LD _="/Process/Bus1/Device_0/Variable_0"/>`
	`<MUL _="/Process/Bus1/Device_0/Variable_1"/>`
	or
	`<LD _="/Process/Bus1/Device_0/Variable_0"/>`
	`<LD _="/Process/Bus1/Device_0/Variable_1"/>`
	`<MUL/>`
	Result: 5332114
	
	Multiplies the signed word values Variable_0=1234 and Variable_2=-4321 of a PLC and loads the Result on the top of the stack.
	`<MULI v1="/Process/Bus1/Device_0/Variable_0"`
	`v2="/Process/Bus1/Device_0/Variable_2"/>`
	Result: -5332114

DIV/DIVI/DIVF – division (unsigned values) / (signed values) / (float)	

Syntax	***`<DIV v1="SystemProperty" v2="SystemProperty"/>`***
	`<DIVI v1="SystemProperty" v2="SystemProperty"/>`

Description	Divides v1 by v2 and writes the result to the top of the stack. Value = v1 / v2 If the result is a real number, the value will be displayed as a rounded value, if the orgin values do not have a decimal part.
Elements	*SystemProperty:* Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14). May also be a constant. If the system property is a signed value or the operation may result in a signed value, DIVI must be used.

Example	Devides the unsigned word value Variable_0=4321 by Variable_1=1234 of a PLC and loads the Result on the top of the stack.
	`<DIV v1="/Process/Bus1/Device_0/Variable_0"`
	`v2="/Process/Bus1/Device_0/Variable_1"/>`
	or
	`<LD _="/Process/Bus1/Device_0/Variable_0"/>`
	`<DIV _="/Process/Bus1/Device_0/Variable_1"/>`
	or
	`<LD _="/Process/Bus1/Device_0/Variable_0"/>`
	`<LD _="/Process/Bus1/Device_0/Variable_1"/>`
	`<DIV/>`
	Result: 3
	
	Devides the signed word value Variable_2=-4321.0 by Variable_1=1234.0 of a PLC and loads the Result on the top of the stack.
	`<DIVI v1="/Process/Bus1/Device_0/Variable_2"`
	`v2="/Process/Bus1/Device_0/Variable_1"/>`
	Result: -3.5

POWF – power operator (float)	
Syntax	***`<POWF v1="SystemProperty" v2="SystemProperty"/>`***
Description	Computes the power. v1 = a, v2=b Result: a^b
Elements	*SystemProperty:* Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14). May also be a constant.

Example	Computes the power of 10 using a PLC variable and loads the Result on the top of the stack.
	`<POWF v1="10"`
	`v2="/Process/Bus1/Device_0/Variable_1"/>`
	or
	`<LD _="10"/>`
	`<DIV _="/Process/Bus1/Device_0/Variable_1"/>`

or
```
    <LD _="10"/>
    <LD _="/Process/Bus1/Device_0/Variable_1"/>
    <POWF/>
```

If Variable_1=3 then Result = 10^3 = 1000

6.3.1.5 Time instruction

TIME - Compare Time Span with current Time	
Syntax	**`<TIME _="YYYY/MM/DD,HH:MM:SS-YYYY/MM/DD,HH:MM:SS"/>`**
Description	Compare the current system time with the given time span and loads a 1 on the top of the stack if the current time lies within the given range. The span may be defined by dates, times or both.
Elements	*YYYY: year [1970…2038]* *MM: month [01…12]* *DD: day [01…31]* *HH: hour [00..23]* *MM: minute [00..59]* *SS: second [00..59]*

Example	System time: 2008/12/19,11:48:30
	
	Check the time span between 11:00 and 12:00.
	`<TIME _="11:00:00-12:00:00"/>`
	Result: 1 (TRUE)
	
	Check the time span between 10:00 and 12:00 at `2008/12/20 to 2008/12/21.`
	`<TIME _="2008/12/20,10:00:00-2008/12/21,12:00:00"/>`
	Result: 0 (FALSE)
	
	Check the date between `2008/12/18 to 2008/12/21.`
	`<TIME _="2008/12/18-2008/12/21"/>`
	Result: 1 (TRUE)

6.3.1.6 Power-on/off delay instruction

D_ON / D_OFF – Power-on/off delay (DWORD only)	

Syntax	**`<DM_ON time="X"/>`**
	`<D_OFF time="X"/>`

Description	Waits the defined time to accept a status as true.
Elements	*X: Time to wait* For time parameters, see chapter 3.1.2

Example	If the service button is pressed for at least 10s, the result of the process variable becomes 1.
	`<LD _="/Process/MB/PollButton"/>`
	`<D_ON time="10s"/>`

If the service button is released, after 10s the result of the process variable becomes 1.

```
<LD _="/Process/MB/PollButton"/>
<D_OFF time="10s"/>
```

6.3.1.7 IF instructions

***IFEQ** – IF equal condition (DWORD only)*

Syntax

```
<IFEQ _="SystemProperty"/>
    Instructions
<ELSE/>
    Instructions
<ENDIF/>
```

Description

Instructions will only be processed if the stack value is equal the condition value, otherwise the instructions following the ELSE condition become processed.

Elements

SystemProperty:
Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14). May also be a constant.

Instructions:
List of instructions

Example

If PLC Variable_0 equals 33, FP Gateway digital input P0 will be set; otherwise input P1 will be set.

```
<LD _="/Process/Bus1/Device_0/Variable_0"/>
<IFEQ _="33"/>
   <LD _="1"/>
   <ST _="/Process/MB/IO/Q/P0"/>
<ELSE/>
   <LD _="1"/>
   <ST _="/Process/MB/IO/Q/P1"/>
<ENDIF/>
```

***IFNE** – IF not equal condition (DWORD only)*

Syntax

```
<IFNE _="SystemProperty"/>
    Instructions
<ELSE/>
    Instructions
<ENDIF/>
```

Description

Instructions will only be processed if the stack value is not equal the condition value, otherwise the instructions following the ELSE condition become processed.

Elements

SystemProperty:
Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14). May also be a constant.

Instructions:
List of instructions

Example	If PLC Variable_0 doesn't equal 33, FP Gateway digital input P0 will be set; otherwise input P1 will be set.
	
	`<LD _="/Process/Bus1/Device_0/Variable_0"/>`
	`<IFNE _="33"/>`
	`   <LD _="1"/>`
	`   <ST _="/Process/MB/IO/Q/P0"/>`
	`<ELSE/>`
	`   <LD _="1"/>`
	`   <ST _="/Process/MB/IO/Q/P1"/>`
	`<ENDIF/>`

6.3.1.8 Text parser instruction

MID – text parser (DWORD only)	
Syntax	**`<MID _="String" start="X" length="Y"/>`**
Description	Loads part of a string from position start to length. Note: The result must be a number.
Elements	*String:* String to parse. If the string is a system property, the reference must be made using the reference string: &#xae; (see chapter 3.1.1) *X:* *Start position* 0 first character If X is larger then length of string, parser starts at end of string. *Y:* *text length*

Example	Extract the "hour" out of the FP Gateway "Time" string:
	`<MID _="&#xae;/TIMES/Time" start="0" length="2"/>`
	or
	`<LD start="0"/>`
	`<LD length="2"/>`
	`<MID _="&#xae;/TIMES/Time"/>`

6.3.1.9 Bit mask instruction

MSK – bit mask with logical result (DWORD only)	
Syntax	***`<MSK v1="SystemProperty" v2="mask"/>`***
Description	MSK is used to mask one or several bits of a given byte, word or dword variable. If at least one masked bit is set, the result is 1, otherwise 0.
Elements	*SystemProperty* Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14). *mask:* (decimal value) *value of the mask.* 1 mask for bit 1 2 mask for bit 2 3 mask for bit 1 OR 2 4 mask for bit 3 5 mask for bit 1 OR 3

	etc…

Example	Masks the unsigned word value Variable_1=1234 of a PLC by mask 7 and loads the logical result on the top of the stack.
	`<MSK v1="/Process/Bus1/Device_0/Variable_1" v2="7"/>`
	
	1234= 00000000000000000000010011010010
	7 = 00000000000000000000000000000111
	
	`Result = 1`

DMSK – bit mask with binary result (DWORD only)	
Syntax	***<DMSK v1="SystemProperty" v2="mask"/>***
Description	DMSK is used to mask one or several bits of a given byte, word or dword variable. The result of the operation is the sum of all bits set within the mask (same as DAND).
Elements	*SystemProperty:* Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14). May also be a constant.

Example	Masks the unsigned word value Variable_1=1234 of a PLC by mask 255 and loads the binary result on the top of the stack.
	`<MSK v1="/Process/Bus1/Device_0/Variable_1" v2="256"/>`
	
	1234= 00000000000000000000010011010010
	255 = 00000000000000000000000011111111
	
	`Result = 210`

6.3.1.10 FIND_BIT_ADDRESS instruction

In most projects an alarm is triggered by a bit variable. Because of the FP Gateway limitation of 100 EventStates (= alarm trigger), only 100 different alarms would be possible. On larger PLC systems, much more alarms are required. These systems may offer the alarm bits within word or dword variables.

With the MSK instruction (see chapter 6.3.1.9), it is possible to trigger an alarm if any of the bits within a word / dword is set, but it isn't possible to select a message text depending on the bit.

FIND_BIT_ADDRESS offers following functionality:

- Up to 7 bytes / words / dwords are composed in a certain order and the FP Gateway counts the bits starting with the lowest bit of the byte / word / dword on the top of the stack. With that a number (bit address) can be assigned to every alarm bit.
- In this list, of bits several bits can be set at the same moment.
- From all simultaneously set bits, only *the first three* will be detected. The bits are counted *starting by 1.*
- The alarm message text is selected via reference depending on the bit address.

FIND_BIT_ADDRESS – Find the address of the n'th bit set (DWORD only)

Syntax

<FIND_BIT_ADDRESS _="*BitRank*" range="*NumberOfStackEntriesToScan*" mask="*CountMask*"/>

Description

Beginning with the entry from the *top of the stack (last loaded value!)* search the '*BitRank'*th bit set in the series of the following '*NumberOfStackEntriesToScan'* values of the stack. Count only the bits which are set in the *CountMask* starting by 1. If such a bit is found, remove all instruction values from the stack and write the counter result at the top of the stack, otherwise write a 0 at the top of the stack.

Elements

BitRank:
First, second, third…bit to find (1… 4294967295).

NumberOfStackEntriesToScan:
Number of values to scan (1….7)

Attention:
Before you call the "Find_Bit_Address" instruction, the specified number of variables must be loaded on the stack, otherwise the instruction results in an error.

CountMask:
Masks the bits to count in a stack value.
value of the mask.

1	mask for bit 1
2	mask for bit 2
3	mask for bit 1 OR 2
4	mask for bit 3
5	mask for bit 1 OR 3

etc…

Example

Define two process variables.
Load Variable_0=1234 and Variable_1=4321 from a PLC and find the first bit set (according to the mask) in the first process variable and the second bit set in the second process variable.

```
<Alarm_0_ProcVar>
   <Value>
      <LD _="/Process/Bus1/Device_0/Variable_1"/>
      <LD _="/Process/Bus1/Device_0/Variable_0"/>
      <FIND_BIT_ADDRESS _="1" range="2" mask="64302"/>
   </Value>
</Alarm_0_ProcVar>

<Alarm_1_ProcVar>
   <Value>
      <LD _="/Process/Bus1/Device_0/Variable_1"/>
      <LD _="/Process/Bus1/Device_0/Variable_0"/>
      <FIND_BIT_ADDRESS _="2" range="2" mask="64302" />
   </Value>
</Alarm_1_ProcVar>
```

Variable_0 is the variable on the top of the stack, because it is loaded after Variable_1.

Using the mask 64302 we get following bit addresses:

Mask 64302:	1	1	1	1	1	0	1	1	0	0	1	0	1	1	1	0
Variable_0:	11	10	09	08	07	–	06	05	–	–	04	–	03	02	**01**	–
Variable_1:	22	21	20	19	18	–	17	16	–	–	**15**	–	14	13	12	–
Variable Values:																
Variable_0 =	0	0	0	0	0	1	0	0	1	1	0	1	0	0	**1**	0
Variable_1 =	0	0	0	1	0	0	0	0	1	1	**1**	0	0	0	0	1

Result:
Alarm_0_ProcVar = 1 (first set bit within mask)
Alarm_1_ProcVar = 15 (second set bit within mask)

6.3.1.11 STRLEN instruction

STRLEN – return the length of a system property	
Syntax	**`<STRLEN _="SystemProperty" />`**
Description	Computes the number of characters within the given system property.
Elements	*SystemProperty* Path to a system property, e.g. I/O, PLC variable, database entry or system variable (see chapter 14). May also be a constant.

Example	Computes the length of the system property SerialNo:
	
	`<Alarm_0_ProcVar>`
	`   <Value>`
	`      <STRLEN _="/SerialNo"/>`
	`   </Value>`
	`</Alarm_0_ProcVar >`
	
	Result: 8

6.3.1.12 FORTH instruction

FORTH – FORTH instruction (DWORD only)	
Syntax	**`<FORTH _="Instruction"/>`**
Description	Offers the possibility to use FORTH instructions. See http://www.forth.org/ for information about the FORTH language.
Elements	*Instruction:* FORTH source code

Example	Convertion of the GPS NMEA longitude "1316114.990234" into WGS84 format:
	`<Alarm_0_ProcVar format="F9.5">`
	`   <Value>`
	`      <LD _="/Process/Bus1/Device_0/Variable_0"/>`
	`      <FORTH _="drop 1 100000 */mod 100000 * swap 100 *`
	`       60 / + 0"/>`
	`   </Value>`
	`</Alarm_0_ProcVar >`
	
	Result: 13.26856

6.3.2 RPN Error Codes

The "Get" command together with the `AddInfo="Error"` attribute can be used to read the error state of a process variable (see chapter 2.4.6.5).

Following error codes may occur:

Code	Description
-217	the interpreter encountered a syntax error
-218	error - stack underflow
-219	error converting number (maybe undefined)
-220	error reading value (group/key not found)
-221	error writinging value (group/key not found, ReadOnly)
-222	stack overflow
-223	variable exists, but is undefined

6.4 Access I/Os and Variables

Most variables of the system properties "/Process/" tree can be read and even set by the Get and Set commands. You can also refer to variables in the message text or any other part of the TiXML project code.

6.4.1 Refer to variable values

You can refer to the current state of the I/Os or the value of a ProcessVariable in a message text or most other project code elements by inserting an appropriate reference. Because these variables are part of the system state, they are also part of the system properties.

See chapter 3.1.1 and 3.8 for further information.

You can also use variables to change connections between configuration elements, e.g. a variable may be used to change a MessageJobTemplate recipient path to select different recipients (Contact_0 or Contact_1) depending on the variable value:

```
<Alarm_0 _="SMTP">
   <Recipient _="/D/AddressBook/Contact_&#xae;/Process/MB/IO/I/P0;"/>
   <Sender _="/D/AddressBook/MySelf"/>
   <Body _="/UserTemplates/Message_0/Body"/>
   <Subject path="/UserTemplates/Message_0/Subject"/>
</Alarm_0>
```

This may also be used to select alarm messages in different languages, different websites and much more.

6.4.2 Read variable values

Using TiXML, you can read the current state of an I/O or variable by the Get command.
See chapter 2.4.6.5 for further information.

Chapter 13 contains an overview of the correct I/O addresses according to the different FP Gateways.

Note:
An open FP Gateway input is indicated by a "1", a closed input port by a "0".

6.4.3 Set outputs, Process- and PLC Variables

Output ports and independent ProcessVariables can be set by the Set command. See chapter 2.4.6.5 for further information.

Set Output Port	
Syntax	**`<Set _="/Process/Address/PortType/Port" value="Value"/>`**
Description	Set the status of the output port addressed by the PortAddress to the given value. See chapter 13 for supported addresses.
Elements	*Address:* Address to the output hardware MB/IO mainboard C40…C4E extension modules *PortType:* Addressing scheme (bit / byte / word / dword): Port Type Range Q 0,1 QB 0…255 QW 0… 65.535 QD 0… 4.294.967.295 *Port:* Address of the port P0…P12 *Value:* Value to set. (0=open or 1=closed contacts)

Example	Close the contacts of relay P2 of the FP Gateway mainboard.
	`<Set _="/Process/MB/IO/Q/P2" value="1"/>`
	
	Open the contacts of output P2 of the extension module with address C42.
	`<Set _="/Process/C42/Q/P2" value="0"/>`
	
	Close the contacts of the first and the second port of the module with the address C40 and open the ports with the index P2-P7.
	`<Set _="/Process/C40/QB/P0" value="3"/>`

Set ProcessVariable	
Syntax	**`<Set _="Address" value="Value"/>`**
Description	Set the value of the ProcessVariable addressed by "Address".
Elements	*Address:* Path to a process variable within the /Process/PV/ tree of the system properties. *Value:* Value to set. (string or number)

Example	Set the ProcessVariable "Alarm_0_ProcVar" to value "23":
	`<Set _="/Process/PV/Alarm_0_ProcVar" value="23"/>`
	
	Set the ProcessVariable "Shiftworker" to value "Contact_0":
	`<Set _="/Process/PV/Shiftworker" value="Contact_0"/>`

Set PLC Variable	
Syntax	**`<Set _="Address" value="Value"/>`**
Description	Set the value of the PLC Variable addressed by "Address".
Elements	*Address:* Path to a PLC variable within the /Process/ tree of the system properties. *Value:* Value to set. (string or number)
Example	Set the PLC Variable "Variable_0" from "Device_0" at "Bus1" to value "23": `<Set _="/Process/Bus1/Device_0/Variable_0" value="23"/>`

6.5 Variable data types and formats

6.5.1 Variable data types

Following data types (attribute "simpleType") are used within the FP Gateway variable processing:

simpleType	Description
Uint8	unsigned 8 Bit value (0...255)
Uint16	unsigned 16 Bit value (0...65535)
Uint32	unsigned 32 Bit value (0...4294967295)
Int8	signed 8 Bit value (-128...+127)
Int16	signed 16 Bit value (-32768...32767)
Int32	signed 32 Bit value (-2147483648...2147483647)
String	text (0...n characters)
Bit	digital value (0...1)
Float	Floating point single precision (±3.402823466*10^{38})
Double	Floating point double precision (±1.7976931348623158*10^{308})

6.5.2 Variable data formats

Without any formatting, the value of I/Os, Process-, System- and PLC variables will be shown natively. The FP Gateway is able to reformat the displayed value into a number format, to replace the status of a Boolean variable with a string and much more. The reformatted value will be used within the text processing (chapter 3.8) and for "Get" commands. Datalogging and Webserver have their own format attributes; the format of the External and ProcessVars database has no function there.

Example

```
<Variable_0 _="I" ind="1" acc="R" format="R10F+4,2;&#xb0;C"/>
```

Variable_0 is an integer value "1234", but with the format string, the displayed value will look like this:

```
<Get _="/Process/Bus1/Device_0/Variable_0"/>
<Get _="+12,34°C"/>
```

Formatting display value of variables

Syntax

On PLC variable definition:

```
<Variable ... simpleType="Uint8" exp="2"
format="Elements;Text"/>
```

On process variable definition:

```
<ProcessVariable format="Elements;Text"/>
```

On Datalogging Record:

```
<ValueName _="Type" size="Length"
format="Elements;Text"/>
```

On query of variable value:

```
<Get _="/Process/PV/Alarm_0_ProcVar"
format="Elements;Text"/>
```

Description

The parameter `format` persists of two parts separated by **semicolon**:

1st part: "Elements"

Contains **Format-Elements** to describe the in- and output of values. Except thousand limiter "T" and number format "F" the format elements can not be combined. The position of the thousand delimiter within format instruction can be choosen at will. The format depends on the type of variable. Not every format is available for all types of variables. The availability of a format element depends on the attribute "simpleType" of the variablen definition. Therefore the valid basic types are given within this discription. The first part may be left empty to show the native values.

2nd part: "Text" (optional)

Contains a **Text** to be displayed together with the value. Within this text, the value may be displayed using its given format of part 1. The position of the value is defined by %%, or will be displayed at the beginning, if %% is left. For some variables, additional values (e.g. physical medium and unit) may be included. The second part may be left empty too. In this case no semicolon is necessary.

Example

Element and Text:	`"T’F+9,2 ;Radius %% cm"`
Only Element:	`"R16"`
Only Text:	`";Text with:%% as value"`

A conditional format allows several sets of both parts `"Elements;Text"` depending on the variable value:

Syntax:

```
{=Condition1}Format1{=Condition2}Format2...{=ConditionN}F
ormatN{}DefaultFormat
```

The curly brackets enclose the condition value to be compared with the variable value. If the variable value is equal the first condition value, the subsequent format instruction will be used. Otherwise, the next condition value will be compared. If no condition matches the variable value, the default format (empty condition "{}") will be used.

Note:
The Text part of the condition format requires %% to define the position of the value. If %% is not specified, the value will not be displayed.

Example

```
"{=1.28}F.1;%%kW{=2.00};-.-{}F"
```

with variable value 1.28: `1.2kW`
with variable value 2.00: `-.-`
with variable value 1.18: `1.18`

	Format elements (Part 1):
? - logical alternative:	?string1,string2 This command is used to replace value by two predefined strings. If the variable is not zero, string1 is displayed, otherwise string2 (boolean format). *Available for following simpleType values:* Bit, Uint8, Uint16, Uint32, Int8, Int16, Int32

Example	**`<Variable_0 simpleType="Uint8"`** […] **`format="?open,closed"/>`**
	
	`<Get _="/Process/Bus1/Device_0/Variable"/>`
	
	FP Gateway answers:
	`<Get _="open"/>` on value 1

* - case alternative:	*Value1:Text1*Value2:Text2**:Text3 This command is used to replace a value by several predefined strings. If the variable value is equal "Value1", "Text1" is displayed, if the variable value is equals "Value2", "Text2" is displayed etc; on every other value, Text3 is displayed. * separator for values to detect (The number of values is not limited.) ** separator for all other values *Available for following simpleType values:* Uint8, Uint16, Uint32, Int8, Int16, Int32 (with exp="0" only)
Example	**`<Variable_0 simpleType="Uint8" exp="0"`** […] **`format="*0:low*1:medium*2:high**:faulty"/>`** `<Get _="/Process/Bus1/Device_0/Variable_0"/>` FP Gateway answers: `<Get _="low"/>` on value 0 `<Get _="medium"/>` on value 1 `<Get _="high"/>` on value 2 `<Get _="faulty"/>` on value 7

R/r - Basis:	Rn/rn
	This command defines the basis n of the value.
	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)

The upper/lower case display of hex letters (A-F) can be specified:

R	Only upper case letters (e.g. AE03)
r	Only lower case letters (e.g. ae03)

Available for following simpleType values:
Uint8, Uint16, Uint32, Int8, Int16, Int32 (with exp="0" only)

Example

HEX:

<Variable_0 simpleType="Uint8" exp="0" [...] **format="R16"/>**

```
<Get _="/Process/Bus1/Device_0/Variable_0"/>
```

FP Gateway answers (variable value=90):
```
<Get _="5A"/>
```

Binary:

<Variable_0 simpleType="Uint8" exp="0" [...] **format="R2"/>**

```
<Get _="/Process/Bus1/Device_0/Variable_0"/>
```

FP Gateway answers (variable value=90):
```
<Get _="1011010"/>
```

T – thousand delimiter:

T*n*
Defines the thousand delimiter.

n= ,	comma (e.g. 12,345,678)
n= .	dor (e.g. 12.345.678)
n= '	apostrophe (e.g. 12'345'678)
n= [empty]	no thousand delimiter (default)

Note:
Can be used in combination with number format element "F".

Available for following simpleType values:
Uint8, Uint16, Uint32, Int8, Int16, Int32, Float, Double

Example

<Variable_0 simpleType="Uint32" [...] **format="T."/>**

```
<Get _="/Process/Bus1/Device_0/Variable_0"/>
```

FP Gateway answers (variable value=98765):
```
<Get _="98.765"/>
```

F - number format

This command defines the format of the number value.
It includes several subitems, which have to be in the given order:

F "sign" "padding" "field width" "decimal point" "fixed point numbers"

sign:
Defines if a sign should be displayed.

+	the sign is displayed, even if the value is positive (e.g. "+12.3", "-12.3")
-	the sign is only displayed, if the value is negative (e.g. "12.3", "-12.3")

	empty the value is unsigned **padding:** Defines how empty positions have to be filled (requires "field width") **0** empty positions are filled with zeros (e.g. 0066.3) **empty** empty positions are cut off (e.g. 66.3) **field width:** Maximum size of the number value output, *including* sign, thousand delimiter, decimal point and the value itself. If omitted, the field width is not limited (and no insertion of padding characters takes place). *Always define enough characters, otherwise the value will be cut off on the left side!* **decimal point:** Character used as decimal separator (option) **,** a comma is used as decimal separator **.** a dot is used as decimal separator (default) **fixed point numbers:** Number of digits behind the decimal separator. Can be omitted, if no decimal point separator is given. **Note:** Can be used in combination with thousand delimiter format element "T". *Available for following simpleType values:* Uint8, Uint16, Uint32, Int8, Int16, Int32, Float, Double

Example	sign:
	`<Variable_0 simpleType="Int16" […] format="F+"/>`
	
	`<Get _="/Process/Bus1/Device_0/Variable_0"/>`
	
	FP Gateway answers (Value = 12345):
	`<Get _="+12345"/>`
	
	Padding, field width:
	`<Variable_0 simpleType="Uint32" exp="-3"[…] format="F09"/>`
	
	`<Get _="/Process/Bus1/Device_0/Variable_0"/>`
	
	FP Gateway answers (Value = 123456):
	`<Get _="00123.456"/>`
	
	Thousand delimiter, sign, field width, decimal point, fixed point numbers:
	`<Variable_0 simpleType="Int32" […] format="T'F+9,2"/>`
	
	`<Get _="/Process/Bus1/Device_0/Variable_0"/>`
	
	FP Gateway answers (Value = 123456):
	`<Get _="+1'234,56"/>`

S – string format	*Sn* Defines the length of a string value. N number of characters to display

	Available for following simpleType values: String
Example	**<Variable_0 simpleType="String"** [...] **format="S3"/>** <Get _="/Process/Bus1/Device_0/Variable_0"/> FP Gateway answers (variable value=ABCDEFG): <Get _="ABC"/>
	Text (part 2)
%%	*Defines the position of the value within the output string.This part is available for all types of data. It is the only format option for data type "String".*
Example	**<Variable_0 simpleType="Int32" exp="-2"** [...] **format="F+;Temp: %%K"/>** <Get _="/Process/Bus1/Device_0/Variable_0"/> FP Gateway answers (Value = 12345): <Get _="Temp: +123.45K"/>
%M% - M-Bus Medium (VIF) %U% - M-Bus Unit (VIF)	Adds the M-Bus (Meterbus) Value Information Field data to the displayed value.
Example	**<Variable_0 simpleType="meterbus" exp="-2"** [...] **format=";Medium:%M% value=%% %U%"/>** <Get _="/Process/Bus1/Device_0/Variable_0"/> FP Gateway answers (Variable value=2530, Heat volume flow): <Get _="Medium:Heat value=25.30 Volume Flow [l/h]"/>

6.6 Analog input

Some FP Gateways offer an analog input 0-10V (12bit).
To convert the value (0-4095, 10V=3798) corresponding to the measured voltage, the "periphery" configuration can be used, which is part of the PROCCFG database. Without this configuration the device automatically converts the values from 0-10000 (10V=10000).

Database path: /PROCCFG/Periphery

Periphery – Analog input	

Syntax	**<Periphery>**
	   <Module Name="ADC 1*12bit" Address="*Module*">
	      <Numerator _="*Numerator*"/>
	      <Denominator _="*Denominator*"/>
	      <Tolerance _="*Tolerance*"/>
	      <Rate _="*Rate*"/>
	   </Module>
	</Periphery>

Description	Configuration of the analog input to convert the measured value.

Elements

Module:
Interface address, e.g. analog input on mainboard: "C9a"

Numerator:
Number on top of the fraction to be multiplied by the measured value.

Denominator:
Number on bottom of the fraction to be multiplied by the measured value (must be >0).

Tolerance:
Changes to be ignored by the analog input relative to converted value. (Default 50)

Rate:
Sample rate to refresh the analog input value in ms. (Default 1000)

Example

a) Display 10 at 10V
If you want to get a range 0-10 (10V=10), there are two solutions:

Periphery
Use the periphery to adjust the range: 10V = 3798*(10/3798)

```
[<SetConfig _="PROCCFG" ver="v">
   <Periphery>
      <Module Name="ADC 1*12bit" Address="C9a">
      <Numerator _="10"/>
      <Denominator _="3798"/>
      <Tolerance _="1"/>
      <Rate _="1000"/>
      </Module>
   </Periphery>
</SetConfig>]
```

ProcessVar (see chapter 6.3)
OR load the AI into a process variable to define decimal places using the "format" option (10V = 10,000):

```
[<SetConfig _="PROCCFG" ver="v">
   <ProcessVars>
      <AI format="F,3">
         <Value>
            <LD _="/Process/MB/A/AI/P0"/>
         </Value>
      </AI>
   </ProcessVars>
</SetConfig>]
```

b) Display 30 at 10V
If you want to get a range 0-30 (10V=30), there are two solutions:

Periphery
Use the periphery to adjust the range: 10V = 3798*(30/3798)

```
[<SetConfig _="PROCCFG" ver="v">
   <Periphery>
      <Module Name="ADC 1*12bit" Address="C9a">
         <Numerator _="30"/>
         <Denominator _="3798"/>
          <Tolerance _="1"/>
         <Rate _="1000"/>
      </Module>
   </Periphery>
</SetConfig>]
```

ProcessVar (see chapter 6.3)
OR load the AI into a process variable and use math operations for the calculation (10V = 10000/1000*3)

```
[<SetConfig _="PROCCFG" ver="v">
   <ProcessVars>
      <AI>
         <Value>
            <LD _="/Process/MB/A/AI/P0"/>
            <DIV _="1000"/>
            <MUL _="3"/>
         </Value>
      </AI>
   </ProcessVars>
</SetConfig>]
```

c) **Display 500 at 3V**
If you want to get a range 0-500 (3V=500), there are two solutions:

Periphery
Use the periphery to adjust the range:
3V = 3798*(3/10)*(500/(3798*(3/10))) = 1139,4*(500/1139,4) = 11394*(5000/11394)

```
[<SetConfig _="PROCCFG" ver="v">
   <Periphery>
      <Module Name="ADC 1*12bit" Address="C9a">
         <Numerator _="5000"/>
         <Denominator _="11394"/>
         <Tolerance _="1"/>
         <Rate _="1000"/>
      </Module>
   </Periphery>
</SetConfig>]
```

ProcessVar (see chapter 6.3)
OR load the AI into a process variable and use math operations for the calculation 3V = 500 = 3000/6

```
[<SetConfig _="PROCCFG" ver="v">
   <ProcessVars>
      <AI>
         <Value>
            <LD _="/Process/MB/A/AI/P0"/>
            <DIV _="6"/>
         </Value>
      </AI>
   </ProcessVars>
</SetConfig>]
```

6.7 S0 Interface

Some FP Gateways offer three S0 interfaces which are used to count pulses as defined in the S0-interface standard. Detailed information regarding the S0 interfaces can be found within the manuals of the FP S ENGuard product family.

6.8 Signal LED

On the front of the Hx400 and Hx600 FP Gateways a "Signal" LED can be found. This LED can be set to different colors and/or flash cycles manually or by event.

Variable path: /Process/MB/SignalLED

Color	Value	Status
Cyclic		
None	0	off
🔴	1	on
	2	blinking (200 ms on, 200 ms off)
	3	blinking (50 ms on, 50 ms off)
	4	blinking (200 ms on, 600 ms off)
	5	blinking (200 ms off, 600 ms on)
	6	blinking 2 time (50ms on, 50ms off) and 600ms off
	7	Flash once every 3 sec
	8	DoubleFlash every 3 sec
🟢	9	on
	10	blinking (200 ms on, 200 ms off)
	11	blinking (50 ms on, 50 ms off)
	12	blinking (200 ms on, 600 ms off)
	13	blinking (200 ms off, 600 ms on)
	14	blinking 2 time (50ms on, 50ms off) and 600ms off
	15	Flash once every 3 sec
	16	DoubleFlash every 3 sec
🔴 🟢	17	blinking 2 time (50ms green, 50ms off, 50ms red, 50ms off) and 600ms off
	18	blinking (200 ms green, 600 ms red)
	19	blinking (200 ms red, 600 ms green)
	20	blinking (200 ms green, 200 ms off, 200 ms red, 200 ms off)
Once		
🔴	21	blinking 1 time
	22	blinking for 4 sec (200 ms on, 200 ms off)
	23	blinking 2 time (50ms on, 50ms off)
🟢	24	blinking 1 time
	25	blinking for 4 sec (200 ms on, 200 ms off)
	26	blinking 2 time (50ms on, 50ms off)
🔴 🟢	27	blinking 1 time

Cyclic LED states (0-20) are kept after reset.

Example

Command to set the signal LED to fast green flashing:
```
[<Set _="/Process/MB/SignalLED" value="14"/>]
```

7. Scheduler

The scheduler can be used to create events at predefined times. These events may be sending status messages or live signals, changing variables or changing database entries e.g. address book entries for shift plans.

7.1 Configuration

Database path: /SCHEDULE/Schedule

```
<Schedule>

    <Time1 _="Event">
        <Weekday _="Mo,Th"/>
        <Time _="19:00"/>
        <Month not="Jan"/>
    </Time1>

    <Timer4 _="Event30Min">
        <Minute _="0,30"/>
    </Timer4>

</Schedule>
```

Scheduler configuration

Scheduler Configuration	

Syntax	***<ScheduleName _="Event">***
	    <ScheduleTimes/>
	      ...
	</ScheduleName>

Description	Attribute group which defines the times of the scheduled event. If several schedulers are using the same point of time, the EventHandler are triggered in the order of the schedulers (from top to bottom).
Elements	*ScheduleName:* Name of the schedule. *Event:* Name of the event, triggered when the schedule time is reached. *ScheduleTimes* List of Attributes describing the times when the event is triggered. (see *times configuration*)

Example	Scheduler configuration for the "Alarm_0" and "Datalogging_0_Log" event.
	The alarm message will be sent each Monday and Thursday at 19:00 but not in January.
	The Datalogging_0_Log writes the current port status every 30 minutes into a logfile.
	
	`<Schedule>`
	    `<Time1 _="Alarm_0">`
	        `<Weekday _="Mo,Th"/>`
	        `<Time _="19:00"/>`
	        `<Month not="Jan"/>`
	    `</Time1>`
	    `<Timer4 _="Datalogging_0_Log">`

```
            <Minute _="0,30"/>
        </Timer4>
    </Schedule>
```

7.2 Time parameters

Times may be configured as periods, e.g. "start-end" or as enumeration e.g. "time1,time2,time3". It is possible to exclude times using "not=" instead of "_=".

Minute	
Description	Minute inside an hour. Valid attributes: **"0,1,2,3,...,59"**

Example	Every quarter of an hour
	`<Minute _="0,15,30,45"/>`
	
	Every minute
	`<Minute _="0-59"/>`

Hour	
Description	Hour inside a day. Valid attributes: **"0,1,2,3,...,23"**
Example	Working hours `<Hour _="9-17"/>`
Time	
Description	Exact times in format "**h:mm**". Valid attributes: **h: "0,1,2,3...,23"** **mm: "00,01,02,...,59"**
Example	at 8:55 and 13:00 `<Time _="8:55,13:00"/>`
Data	
Description	Exact Date in format "**d.m.[yyyy]**". Year is option. Valid attributes: **d: "1,2,3,...,31"** **m: "1,2,3,...,12"** **yyyy:"1970,1971,1972,...,2038"**
Example	no german holidays `<Data not="1.1.,1.5.,3.10.,24.12.-26.12.,31.12." />`
Day	
Description	Day valid for all month. Valid attributes: **"1,2,3,...,31"**
Example	First five days of each month <Day _="1-5"/>
Month	
Description	Month valid for all years. Valid attributes: **"Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec"** or "**1**"-"**12**".
	February, June and October <Month _="Feb,6,Oct"/>

Example	
Weekday	
Description	Weekday valid for all weeks. Valid attributes: **"Mo, Tu, We, Th, Fr, Sa, Su"** or "**0**" **(=Su) – "6" (=Sa)**.
Example	Workdays only `<Weekday _="Mo-Fr"/>`
Condition	
Description	Condition is used to refer to other times within the /SCHEDULE/condition database.
Example	`<Condition _="Condition/FiveMinutes"/>`

Example

```
[<SetConfig _="SCHEDULE" ver="y">
  <Condition>
    <FiveMinutes>
      <Minute _="0,5,10,15,20,25,30,35,40,45,50,55"/>
    </FiveMinutes>
    <Holidays>
      <Data _="1.1.,1.5.,3.10.,25.12.,26.12."/>
    </Holidays>
    <NoHoliday>
      <Data not="1.1.,1.5.,3.10.,25.12.,26.12."/>
    </NoHoliday>
  </Condition>
</SetConfig>]

[<SetConfig _="SCHEDULE" ver="y">
  <Schedule>
    <!-- every Monday and Thursday at 19:00, but not in January -->
    <Time1 _="Event">
      <Weekday _="Mo,Th"/>
      <Time _="19:00"/>
      <Month not="Jan"/>
    </Time1>
    <!—Monday-Friday at 0800-2000 all 5 minutes but not at holidays -->
    <Time2 _="Event">
      <Weekday not="Sa,Su"/>
      <Hour _="8-20"/>
      <Condition _="Condition/Fiveminutes"/>
      <Condition not="Condition/Holidays"
    </Time2>
    <!--Monday-Friday 0740 to 1640 all 20 minutes but not at holidays-->
    <Time3 _="Event">
      <Weekday _="Mo-Fr"/>
      <Time _="7:40-16:40"/>
      <Minute _="0,20,40"/>
      <Condition _="Condition/NoHoliday"/>
    </Time3>
    <!-- all 30 minutes -->
    <Timer4 _="Event30Min">
      <Minute _="0,30"/>
    </Timer4>
  </Schedule>
</SetConfig>]
```

7.3 ScheduleDefinition

To prevent upload problems of the SCHEDULE database groups (conditions for schedulers missing - vice versa), the SCHEDULE database was redesigned. We recommend to use the new structure even if the old format is still supported and used by TILA. Old and new structure must not be used in conjunction.
The "Schedule" and the "Condition" group are now both part of the "ScheduleDefinition" group inside SCHEDULE database. The structure inside both groups didn't change. Refer to chapters 7.1 7.1 and 7.2 for more information.

Database path: /SCHEDULE/ScheduleDefinition

Example

```
[<SetConfig _="SCHEDULE" ver="y">
  <ScheduleDefinition>
    <Schedule>
      <Time2 _="Event">
        <Weekday not="Sa,Su"/>
        <Hour _="8-20"/>
        <Condition _="Condition/Fiveminutes"/>
      </Time2>
    </Schedule>
    <Condition>
      <FiveMinutes>
        <Minute _="0,5,10,15,20,25,30,35,40,45,50,55"/>
      </FiveMinutes>
    </Condition>
  </ScheduleDefinition>
</SetConfig>]
```

7.4 Testing

***ScheduleTest** - Calculates a list of scheduler event times*	
Syntax	<ScheduleTest _="*range*" max="*maxcount*"/>
Description	This command returns a list of calculated scheduler event times in a given time range.
Parameter	*range:* "*Timestamp1-Timestamp2*": scheduled event times between both timestamps "*Timestamp*": scheduled event times from now until timestamp "-*Timestamp*": scheduled event times from now until timestamp "*Timestamp*-": scheduled event times from timestamp to maxcount *Timestamp format:* "DD.MM.YYYY[,hh:mm]" or "YYYY/MM/DD[,hh:mm]" "next *N unit*" scheduled event times from now until next N units Units format: "Hours", "Days", "Months", "Years" *maxcount:* Number of scheduler event times to calculate (optional, default: 100).

Return	*If no error (command is processed):*
	`<ScheduleTest>`
	`   <SE_1 _="2004/03/04,16:10:00">      <Event _="Eventname"/>   </SE_1>   <SE_2 _="2004/03/04,16:10:00">      <Event _="Eventname"/> </SE_2>`
	`    ...`
	`   <SE_100 _="2004/03/06,09:20:00">      <Event _="Eventname"/>   </SE_100>`
	`</ScheduleTest>`
	On error (command is not processed):
	see default error frame (chapter 2.4.4)

Example	All scheduled event times of the next 14 days, max 100 entries:
	`<ScheduleTest _="next 14 Days"/>`
	
	All scheduled event times from now until 31.12.2003, max 25 entries:
	`<ScheduleTest _="-31.12.2003" max="25"/>`
	
	All scheduled event times between 22.8.2003 09:00 and 31.10.2003 22:00, max 100 entries
	`<ScheduleTest _="22.8.2003,9:00-2003/10/31,22:00"/>`
	
	All scheduled event times starting from 1.1.2004, max 50 entries:
	`<ScheduleTest _="1.1.2004-" max="50"/>`

8. Sequencer

The scheduler (previous chapter) enables the device to change values on a point of time. The values have to be part of the set command inside the event handler. To change values more dynamically, a special feature called "sequencer" was implemented.

The sequencer uses different profiles of value lists with points of time to change the values. Each list may have a special priority. The lowest priority (0) will be active always, higher priorities will deactivate the the lower priorities at the given point of time.

The FP Gateway will process these lists sequencially to change the associated variables.

8.1 Configuration

The configuration of the sequencer is part of the SCHEDULE database, group "Sequencer".

Database path: /SCHEDULE/Sequencer

***Sequencer** – Profiles*	

Syntax	**`<Sequencer>`**
	`   <`*`Profilename`*` event="`*`Eventname`*`" logfile="`*`Logfilename`*`"/>`
	`</Sequencer>`

Description	This configuration enables a sequencer profile and associates an event and logfile to it.
Parameter	*Profilename:* The sequencer may have several profiles for different events. Random names are possible but have to be unique throughout the configuration. *Eventname:* Name of the event handler which will process the sequencially changed values. The event handler has to exist inside EventHandler group (database EVENTS). *Logfilename:* Name of the logfile in which a copy of the profile will be stored during "SetSequence" command. The logfile has to exist inside LogDefinition group (database LOG)

Example	Sequencer profile which will call the event "Switch_0". The Profiles will be copied into "Profiles" logfile:
	
	`<Sequencer>`
	`   <Sequence_0 event="Switch_0" logfile="Profiles"/>`
	`</Sequencer>`

8.2 Changing sequences

The sequencer profiles configured in the previous chapter are empty by default. To define sequencer times and values, the SetSequence command is used:

***SetSequence** – Defines sequencer event times and values*	

Syntax	**`<SetSequence _="`*`Profile`*`" priority="`*`n`*`" mode="`*`format`*`"`**
	`  mask="`*`Mask`*`">`
	`   <T date="`*`Date`*`" time="`*`Time`*`" P1="`*`p1`*`" P2="`*`p2`*`" … P6="`*`p6`*`"/>`
	`        …`

	</SetSequence>
Description	This command defines the sequencer event times and values. The values will be processed by the associated event handler at the given point of time.
Parameter	*Profile:*
	Name of profile to be changed. Has to exist inside the Sequencer group.
	n:
	Priority of the sequencer profile. Priority range: 0 (low)-255 (high). Only three different priorities per profile are possible. Priority 0 has to be the basic profile. See next chapter to learn more about profile priorities.
	format:
	The sequencer profiles may be transferred in two different formats:
	XML: Data will be transferred in TiXML syntax (default)
	TEXT: Data will be transferred in TEXT format (for profiles with CSV format)
	In TEXT format the raw data has to be enclosed by a XML frame:
	`<!CDATA[`
	`****Data****`
	`]]>`
	Mask:
	Defines the data format in TEXT mode (not necessary for XML-mode)
	d: Date
	t: Time
	1: Value P1
	…
	6: Value P6
	e.g.
	mask="d;t;1;2;3;4;5;6"
	for data in this format:
	DD.MM.YYYY;hh:mm;P1;P2;P3;P4;P5;P6
	01.01.2004;09:15;200;300;400;500;600
	Date:
	Date for the sequenced time of event.
	Valid formats:
	DD.MM
	DD.MM.YY
	DD.MM.YYYY
	YY/MM/DD
	YYYY/MM/DD
	An asterisk "*" or "0" may be used to replace each unit.
	E.g. "00.03.00" is same as "*.03.*" which means every day in march every year.
	Time:
	Point of time for the sequenced event. Format: hh:mm
	An asterisk "*" may be used to replace each unit.
	p1…p6:

	List of values to be processed by sequenced event handler (max 6). The event handler has to refer to these values via process reference &#xae;~/Px; (see chapter 3.1.1)

Return	***If no error (command is processed):***
	`<SetSequence/>`
	On error (command is not processed):
	see default error frame (chapter 2.4.4)

Example	1. Every day at 09:00 a "minimum" variable (P1) has to be 20, a maximum variable (P2) has to be 80. Every day at 18:00 a "minimum" variable (P1) has to be 30, a maximum variable (P2) has to be 70.
	
	`[<SetSequence _="LevelProfile" priority="0">`
	`   <T date="*.*.*" time="09:00" P1="20" P2="80"/>`
	`   <T date="*.*.*" time="18:00" P1="30" P2="70"/>`
	`</SetSequence>]`
	
	Date "*.*.*" will be processed on each day, every month, every year.
	
	Same Example in TEXT format:
	
	`<SetSequence _="LevelProfile" priority="0" mode="TEXT"`
	`mask="d;t;1;2">`
	`   <![CDATA[`
	`   *.*.*;09:00;20;80`
	`   *.*.*;18:00;30;70`
	`   ]]>`
	`</SetSequence>`
	
	2. Additional to the previous example, both variables should have following values on each day in april:
	
	`[<SetSequence _="LevelProfile" priority="1">`
	`   <T date="*.04.*" time="09:00" P1="15" P2="85"/>`
	`   <T date="*.04.*" time="18:00" P1="25" P2="75"/>`
	`</SetSequence>]`
	
	Date "*.04.*" will be processed on each day, in april, every year. Due to the higher priority, the profile of example 1 will be inactive at the given time.
	
	3. Values change every 15 minutes:
	`[<SetSequence _="PowerProfile" priority="0">`
	`   <T date="*.*.*" time="*:00" P1="10" P2="80"/>`
	`   <T date="*.*.*" time="*:15" P1="20" P2="90"/>`
	`   <T date="*.*.*" time="*:30" P1="30" P2="100"/>`
	`   <T date="*.*.*" time="*:45" P1="20" P2="90"/>`
	`</SetSequence>]`

To delete a sequence, the sequence definition inside sequencer configuration has to be deleted.

A sequence may also be transferred to the device via email.
See chapter 9 for more information.

8.2.1 Profile priorities

For each sequencer profile, a maximum of 3 priorities is allowed.

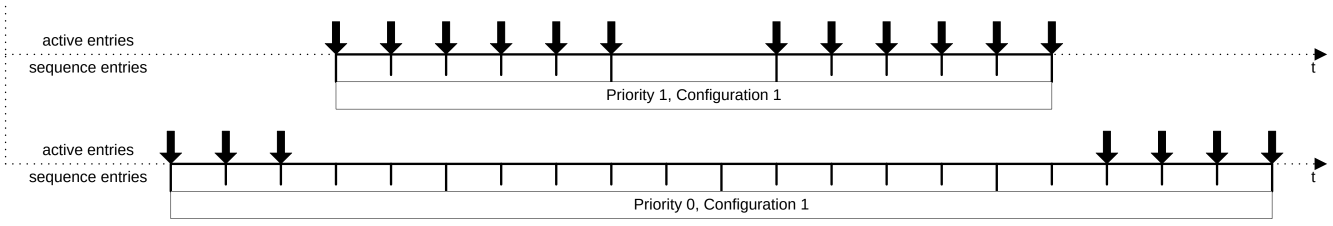
The sequencer handles the profiles in different ways, related to their priority:

Priority 0: A sequence with priority 0 will be **replaced** by a sequence with priority 0.
Priority >0: A sequence with priority >0 will be supplemented by new data with same priority. All expired entries will be deleted.

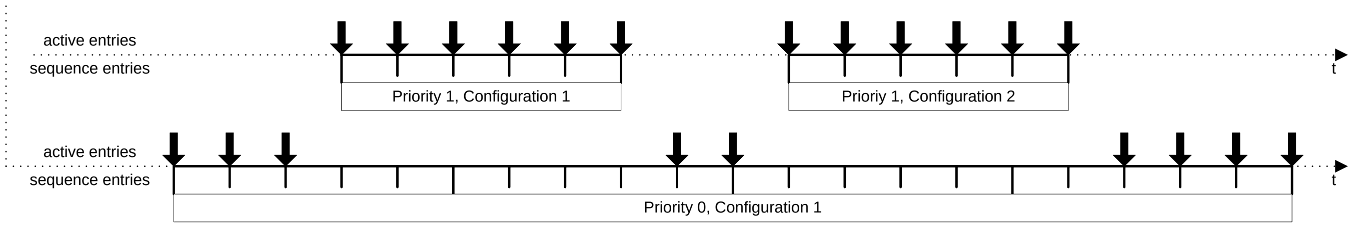
If several sequencer events inside a single profile are configured with the same point of time, the sequence with the highest priority will be processed only.

Examples

A) Between two points of time with higer priority, no lower priorities will be processed:



B) If a sequence with same priority >0 was configured several times, all lower priorities between both sequences will be processed:


active entries
sequence entries
Priority 1, Configuration 1
Prioriy 1, Configuration 2
t
active entries
sequence entries
Priority 0, Configuration 1
t

8.3 Testing

SequenceTest - *Calculates a list sequencer event times*	
Syntax	<SequenceTest _="*Profilename*" range="*Range*" max="*maxcount*"/>
Description	This command returns a list of calculated secuencer event times in a given time range.
Parameter	***Profilename:*** Name of profile to be tested. Has to exist inside Sequencer group. ***Range:*** *`Timestamp1-Timestamp2`*": sequencer event times between both timestamps "*`Timestamp`*": sequencer event times from now until timestamp "-*`Timestamp`*": sequencer event times from now until timestamp "*`Timestamp-`*": sequencer event times from timestamp to maxcount *Timestamp format:* "DD.MM.YYYY[,hh:mm]" or "YYYY/MM/DD[,hh:mm]" `"next N unit"` scheduled event times from now until next N units

Units format: `"Hours"`, `"Days"`, `"Months"`, `"Years"`)

maxcount:
Number of sequencer event times to calculate (default: 100).

Return

If no error (command is processed):

```
<SequenceTest>
   <SEQ_1 _="2004/02/04,08:01:00">      <Event
_="Eventname" P1="4" P2="14" P3="" P4=""
       P5="" P6=""/>   </SEQ_1>   <SEQ_2
_="2004/02/04,08:02:00">      <Event _="Eventname" P1="5"
P2="13" P3="" P4=""
       P5="" P6=""/>   </SEQ_2>
    ...
   <SEQ_100 _="2004/02/04,09:40:00">      <Event
_="Eventname" P1="9" P2="7" P3="" P4=""
       P5="" P6=""/>   </SEQ_100>
</SequenceTest>
```

On error (command is not processed):
see default error frame (see chapter 2.4.4)

Example

All sequencer event times of the next 14 days, max 100 entries:

```
<SequenceTest _="Sequence_0" range="next 14 Days"/>
```

All sequencer event times from now until 31.12.2003, max 25 entries:

```
<SequenceTest _="Sequence_0" range="-31.12.2003" max="25"/>
```

All sequencer event times between 22.8.2003 09:00 and 31.10.2003 22:00, max 100 entries

```
<SequenceTest _="Sequence_0" range="22.8.2003,9:00-
2003/10/31,22:00"/>
```

All sequencer event times starting from 1.1.2004, max 50 entries:

```
<SequenceTest _="Sequence_0" range="1.1.2004-" max="50"/>
```

8.4 Example

Sequencer logfile definition:

```
[<SetConfig _="LOG">
   <LogFiles>
      <Profiles size="125000"/>
   </LogFiles>
</SetConfig>]
```

Sequencer profile definition:

```
[<SetConfig _="SCHEDULE">
   <Sequencer>
      <Sequence_0 event="Switch_0" logfile="Profiles"/>
   </Sequencer>
</SetConfig>]
```

Event Handler to be triggered by sequencer:

```
<Switch_0>
   <Set _="/Process/Bus1/Device_0/Variable_0" value="&#xae;~/P1;"/>
      <Set _="/Process/Bus1/Device_0/Variable_1"
       value="&#xae;~/P2;"/>
</Switch_0>
```

Sequencer times and values:

```
[<SetSequence _="Sequence_0" priority="0">
   <T date="*.*.*" time="*:00" P1="10" P2="80"/>
   <T date="*.*.*" time="*:15" P1="20" P2="90"/>
   <T date="*.*.*" time="*:30" P1="30" P2="100"/>
   <T date="*.*.*" time="*:45" P1="20" P2="90"/>
</SetSequence>]
```

9. Processing incoming messages

9.1 Introduction

An FP Gateway can be controlled by received emails and SMS or by a simple phone call (callerID).

Scenario:
The FP Gateway is connected to a PLC with output ports. At the output ports, some actuators are connected by an electrical signal line (for example a fan).

Password
******
Command
-> ok

The remote control user sends a SMS including a password (*SECRET*) and a command word (*FAN_ON*):

```
SECRET FAN_ON
```

The FP Gateway receives the message and triggers an event with the same name as the command word (in the example 'FAN_ON'). This is handled as if received from a client as an event message (DoOn via Command).

Please note that the command name of the incoming SMS message will be converted into upper case, therefore the EventHandler names must be uppercase, too.

The event is processed according to the commands defined by the corresponding event handler configuration:

Database path: /EVENTS/EventHandler

```
<EventHandler>
   <FAN_ON>
      <Set _="/Process/Bus1/Device_0/Variable_0" value="1"/>
      <SendMail _="MessageJobTemplates/Switch_0"/>
   </FAN_ON>
</EventHandler>
```

The event sets output 'Variable_0' of the connected PLC which starts the connected fan.

Additionally, a message job is started which sends an answer back to the sender of the command, e.g.:

```
OK: FAN_ON
```

9.2 Event via incoming call (callerID)

All FP Gateways are able to detect incoming callerIDs using the CLIP service. If a callerID matches an entry in the callerID database, the call will not be answered by the Device. Instead it will start processing the event assigned to the callerID.

This may be used to open a garage door just by a simple phone call without any costs for instance.

The callerID database is inside the ISP database.

Database path: /ISP/IncomingCallTrigger

Incoming Call Trigger	

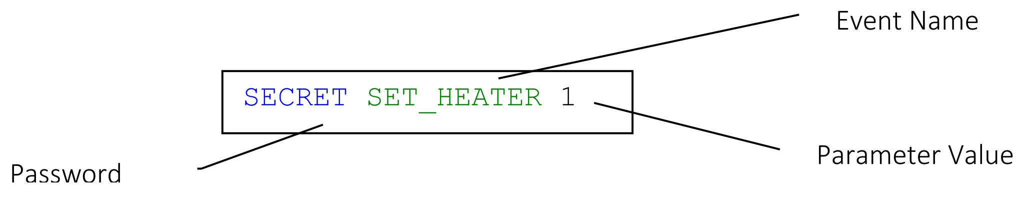
Syntax	**`<IncomingCallTrigger>`**
	`   <NoX _="callerID" event="EventName"/>`
	`</IncomingCallTrigger>`

Description	A list of callerIDs (max. 5) with events which will be processed after this callerID was detected by the FP Gateway.
Elements	*X:* Entry number. Value 1 - 5 *CallerID:* CallerID to be detected by the FP Gateway. Use wildcards "*" to replace a part of the callerID or "?" to replace a single digit. *EventName:* Name of the event to be processed if the callerID was detected.

Example	Mobile phone number +491721234567 will trigger the event "Switch_0", mobile phone numbers +491727654321 and +491727654355 will trigger the event "Switch_1":
	`[<SetConfig _="ISP">`
	`   <IncomingCallTrigger>`
	`      <No1 _="+491721234567" event="Switch_0"/>`
	`      <No2 _="+4917276543*" event="Switch_1"/>`
	`   </IncomingCallTrigger>`
	`</SetConfig>]`
	Note:
	The format of the caller ID depends on the telephone provider. Sometimes the area code is left or "+" is transmitted as "00". Use the Tixi CLIP tool to detect the caller ID format required by the connection of your Device.

9.3 Event via incoming message (SMS, Email)

To map an incoming message to an event, the message (subject) must have a special syntax. Example:



An event handler must be configured accordingly to the submitted Event Name (for SMS always upper case!). Like for DoOn command, additional parameters can be used which can be processed in the event handler, the message job template or the message text template.

The following message format is required.

Incoming Message	
Syntax	***Password* SPACE *EventName* SPACE *Parameter1* SPACE *Parameter2...***
Description	Format of the subject line (email) or content (SMS) of an incoming message to trigger an event.
Elements	***Password:*** Password to access the device. 1...20 characters (not empty), no SPACE character allowed. ***EventName:*** Name of the event to be triggered (for SMS upper case!). There must be an event handler configured for this event. 1...20 characters (not empty), XML tag characters only. ***Parameter1...Parameter10:*** Value of the n'th parameter (no SPACE character, only 29 characters per parameter if using SMS).
Example	User password `SECRET`, trigger the `HEATER_ON` event and submit the parameter 1. `SECRET HEATER_ON 1`

9.3.1 Event parameter generated by an incoming message

Unlike the default events, the event created by incoming messages contains predefined parameters. These parameters can be used to control the FP Gateway or to create an answer message.

The event handling can be tested independently. Therefore, the event messages created by an incoming message can be "simulated" by sending a DoOn message with the events described below. When this test is finished ok, the message to event map could be tested by sending the corresponding incoming messages.

Each message type has its own specific parameter list.

The event generated by incoming message has the following format:

Event created by an incoming message	

Syntax	`<DoOn _="EventName">`
	`   <Event _="EventName"/>`
	`   <Password _="Password"/>`
	`   <Alpha _="SenderAlias"/>`
	`   <OA _="OA"/>`
	`   <Time _="ReceiveTime"/>`
	`   <RemoteSerialNo _="RemoteSerialNo"/>`
	`   <Text _="MessageText"/>`
	`   <P1 _="ValuesOfParameter1"/>`
	`   <P2 _="ValuesOfParameter2"/>`
	`   <P3 _="ValuesOfParameter3"/>`
	`   <P4 _="ValuesOfParameter4"/>`
	`   <P5 _="ValuesOfParameter5"/>`
	`   <P6 _="ValuesOfParameter6"/>`
	`   <P7 _="ValuesOfParameter7"/>`
	`   <P8 _="ValuesOfParameter8"/>`
	`   <P9 _="ValuesOfParameter9"/>`
	`   <P10 _="ValuesOfParameter10"/>`

	`</DoOn>`
Description	Structure of the event created by an incoming message.

Elements	***`EventName:`***
	Event name parsed from received message text.
	`Password:`
	Password parsed from received message text.
	`SenderAlias:`
	Address of the sender or alphanumerical representation of the originating address if any stored in the chip card (SMS) or alias name of sender (email).
	`OA:`
	Originating address (callerID) received from telephone network (SMS) or sender address (from field) parsed from received message header (email).
	`ReceiveTime:`
	Time stamp received from GSM network (SMS) or Time stamp indicating when the message was received (email).
	`MessageText:`
	The text from the 'Subject' line of the received message (email) or content of the message (SMS).
	`P1...P10(optional)`
	Value of the parameter delivered by the message if any.

Example	Event message generated from an incoming SMS of this format: `SECRET HEATER_ON 1`
	`<DoOn ="HEATER_ON">`
	`   <Event _="HEATER_ON"/>`
	`   <Password _="SECRET"/>`
	`   <OA _="+491717959463"/>`
	`   <Time _="01/07/20,08:56:33+08"/>`
	`   <Text _="SECRET HEATER_ON 1"/>`
	`   <Alpha _="CON"/>`
	`   <P1 _="1"/>`
	`</DoOn>`
	Event message generated from a received POP3 email with this subject:
	`SECRET HEATER_ON 1`
	
	`<DoOn ="HEATER_ON">`
	`   <Event _="HEATER_ON"/>`
	`   <Password _="SECRET"/>`
	`   <OA _="tixi-support@inovolabs.com"/>`
	`   <Time _="01/07/2004,08:56:33+08"/>`
	`   <Text _="SECRET HEATER_ON 1"/>`
	`   <Alpha _="tixi-support@inovolabs.com"/>`
	`   <P1 _="1"/>`
	`</DoOn>`

9.3.2 System events for invalid incoming messages

In case the message cannot be processed properly, a predefined event is triggered that allows notification of the fault to be given and enables the tracking of intrusion attempts.

Event created on event processing error

Syntax	**`<DoOn _="System/TixiInvalidEvent">`**
	…
	or
	`<DoOn _="System/SMSInvalidEvent">`
	…
	or
	`<DoOn _="System/POPInvalidEvent">`
	`<Event _="`*`EventName`*`"/>`
	`<Password _="`*`Password`*`"/>`
	`<Alpha _="`*`SenderAlias`*`"/>`
	`<OA _="`*`OA`*`"/>`
	`<Time _="`*`ReceiveTime`*`"/>`
	`<RemoteSerialNo _="`*`RemoteSerialNo`*`"/>`
	`<Text _="`*`MessageText`*`"/>`
	`<P1 _="`*`ValuesOfParameter1`*`"/>`
	`<P2 _="`*`ValuesOfParameter2`*`"/>`
	`<P3 _="`*`ValuesOfParameter3`*`"/>`
	`<P4 _="`*`ValuesOfParameter4`*`"/>`
	`<P5 _="`*`ValuesOfParameter5`*`"/>`
	`<P6 _="`*`ValuesOfParameter6`*`"/>`
	`<P7 _="`*`ValuesOfParameter7`*`"/>`
	`<P8 _="`*`ValuesOfParameter8`*`"/>`
	`<P9 _="`*`ValuesOfParameter9`*`"/>`
	`<P10 _="`*`ValuesOfParameter10`*`"/>`
	`<ErrNo _="`*`ErrNo`*`"/>`
	`<ErrText _="`*`ErrorText`*`"/>`
	`</DoOn>`

Description	Structure of the event created by an incoming message which could not be processed. This event corresponds to a DoOn event message. Note that an event handler of that name must exist. The event parameters depend on the type of message.
Elements	*EventName:* Event name parsed from received message text. *Password:* Password parsed from received message text. *SenderAlias:* Alphanumerical representation of the originating address if any stored in the chip card (SMS) or alias name of sender (email). *OA:* Originating address (callerID) received from telephone network (SMS) or sender address (from field) parsed from received message header (email). *ReceiveTime:* Time stamp received from GSM network (SMS) or time stamp indicating when the message was received (email). *MessageText:* The text from the 'Subject' line of the received message (email) or content of the message (SMS). *P1...P10 (optional)* Value of the parameter delivered by the message if any. *ErrNo:*

	Numerical representation of the processing error. ***ErrorText:*** Textual representation of the processing error.
Example	Error event message generated from an incoming SMS of this format: `SECRET HEATERON 1`

```
<DoOn _="System/SMSInvalidEvent ">
   <Event _="HEATERON"/>
   <Password _="SECRET"/>
   <OA _="+491717959463"/>
   <Time _="01/07/20,08:56:33+08"/>
   <Text _="SECRET HEATERON 1"/>
   <Alpha _="CON"/>
   <ErrNo _="-300"/>
   <ErrText _="Invalid event name"/>
   <P1 _="1"/>
</DoOn>
```

Error event message generated from a received POP3 email with this subject:
`SECRET HEATERON 1`

```
<DoOn _="System/POPInvalidEvent ">
   <Event _="HEATERON"/>
   <Password _="SECRET"/>
   <OA _="tixi-support@inovolabs.com"/>
   <Time _="01/07/20,08:56:33+08"/>
   <Text _="SECRET HEATERON 1"/>
   <Alpha _="InovoLabs Support"/>
   <ErrNo _="-300"/>
   <ErrText _="Invalid event name"/>
   <P1 _="1"/>
</DoOn>
```

In case the message password is wrong, a predefined event is triggered that allows notification of the fault to be given and enables the tracking of intrusion attempts.

Event created on invalid password

Syntax

```
<DoOn _="System/TixiInvalidPassword">
   …
or
<DoOn _="System/SMSInvalidPassword">
   …
or
<DoOn _="System/SMSInvalidPassword">
   <Event _="EventName"/>
   <Password _="Password"/>
   <Alpha _="SenderAlias"/>
   <OA _="OA"/>
   <Time _="ReceiveTime"/>
   <RemoteSerialNo _="RemoteSerialNo"/>
   <RemoteBoxnumber _="RemoteBoxnumber"/>
   <RemoteBoxname _="RemoteBoxname"/>
   <Text _="MessageText"/>
   <P1 _="ValuesOfParameter1"/>
   <P2 _="ValuesOfParameter2"/>
   <P3 _="ValuesOfParameter3"/>
   <P4 _="ValuesOfParameter4"/>
   <P5 _="ValuesOfParameter5"/>
   <P6 _="ValuesOfParameter6"/>
   <P7 _="ValuesOfParameter7"/>
   <P8 _="ValuesOfParameter8"/>
   <P9 _="ValuesOfParameter9"/>
   <P10 _="ValuesOfParameter10"/>
   <ErrNo _="ErrNo"/>
   <ErrText _="ErrorText"/>
</DoOn>
```

Description

Structure of the event created by an incoming message which contains an invalid password. This event corresponds to a DoOn event message. Note that an event handler of that name must exist.

Note:
If no Def_Message user is configured in the AccRights database this system event will always be processed.

Elements

EventName:

Event name parsed from received message text.

Password:

Password parsed from received message text.

SenderAlias:

Address of the sender or alphanumerical representation of the originating address if any stored in the chip card (SMS) or alias name of sender (email).

OA:

Originating address (callerID) received from telephone network (SMS) or sender address (from field) parsed from received message header (email).

ReceiveTime:

Time stamp received from GSM network (SMS) or time stamp indicating when the message was received (email).

MessageText:

The text from the ‘Subject’ line of the received message (email) or content of the message (SMS).

P1...P10(optional)

Value of the parameter delivered by the message if any.

ErrNo:

Numerical representation of the processing error.

ErrorText:

Textual representation of the processing error.

Example

Event message created by an incoming SMS of this format: TRY HEATER_ON 1 in case TRY is not the correct password:

```
<DoOn _="System/SMSInvalidPassword">
   <Event _="HEATER_ON"/>
   <Password _="TRY"/>
   <OA _="+491717959463"/>
   <Time _="01/07/20,08:56:33+08"/>
   <Text _="TRY HEATER_ON 1"/>
   <Alpha _="CON"/>
   <P1 _="1"/>
</DoOn>
```

Event message created by a received POP3 email with this subject:
`TRY HEATER_ON 1` in case `TRY` is not the correct password:

```
<DoOn _="System/POPInvalidPassword">
   <Event _="HEATER_ON"/>
   <Password _="TRY"/>
   <OA _="tixi-support@inovolabs.com"/>
   <Time _="01/07/20,08:56:33+08"/>
   <Text _="TRY HEATER_ON 1"/>
   <Alpha _="InovoLabs Support"/>
   <P1 _="1"/>
</DoOn>
```

9.3.3 Receiving SMS (GSM and PSTN)

To process incoming SMS with a FP Gateway GSM, the SIM card of the device must not contain any SMS.

Several SMS-providers are using different character sets and special characters are converted unexpected. Therefore, we recommend using EventHandler names without special characters.

Receiving SMS on PSTN may not be supported with all telephone providers.

9.3.4 Collecting Internet emails (POP3)

To process incoming POP3 emails, the FP Gateway has to collect them first.

Make sure that the POP3 details are configured in the ISP database (see chapter 4.7).
This simple EventHandler will do the job:

Database path: /EVENTS/EventHandler

```
<POP3>
   <POP3Query/>
</POP3>
```

The FP Gateway will only receive messages with matching password.

A good solution can be implemented by combining the POP3 query event with the scheduler.
Following example will collect emails every 15 minutes.

Database path: /SCHEDULE/Scheduler

```
<POP3 _="POP3">
   <Minute _="0,15,30,45"/>
</POP3>
```

9.3.4.1 Email filter

The FP Gateway is able to filter email messages to shorten online time, skip spam messages and to share a single POP3 account with other FP Gateways.
Therefore, a user defined filter word can be included at the end of the email subject or (if "Lines" are specified) within the message body. See chapter 3.5 for details.

If a filter is specified, the device will ignore all messages without this filter word, and even leave them on the server.

9.3.5 Example

9.3.5.1 Event Handler

The typical application for the use of incoming message is assumed to be the following:

1.Log the message.

2.Set a variable.

3.Send a reply.

As described above, there are a number of events created automatically by an incoming message. Each must be handled by an event handler.

For the error cases, the following actions are assumed to be done:

Invalid Password:

Log the incoming message along with sender address.

Invalid event:

1. Log the incoming message.
2. Send a notification on the problem.

To process the error events, insert a section as follows into the EventHandler group of the EVENT database. The example assumes that the `Switch_0` as well as the `AnswerOnXXXError` message jobs have been configured.

Note:
The `InvalidPassword` and the `InvalidEvent` sections are contained in a special sub-section `SYSTEM` of the `EVENT` database.

Database path: /EVENTS/EventHandler

```
<EventHandler>
   <HEATER_ON>
      <Log _="IncomingMessage">
         <Annotation ="Incoming message"/>
         <Sender _="&#xae;~/Alpha"/>
         <OA _="&#xae;~/OA"/>
         <Time _="&#xae;~/Time"/>
      </Log>
      <Set _="/Process/C42/Q/P4" value="1"/>
      <SendMail _="MessageJobTemplates/Switch_0"/>
   </HEATER_ON>
 <System>
   <TixiInvalidPassword>
      <Log _="FailedIncomingCall"/>
      <Annotation _="ExpressEMail with invalid password received"/>
      <Sender _="&#xae;~/Alpha"/>
      <Time _="&#xae;~/Time"/>
      <Text _="&#xae;~/Text"/>
      </Log>
   </TixiInvalidPassword>
   <TixiInvalidEvent>
      <Log _="FailedIncomingCall"/>
      <Annotation _="Express-E-Mail with invalid event received"/>
      <Sender _="&#xae;~/Alpha"/>
      <Time _="&#xae;~/Time"/>
      <Text _="&#xae;~/Text"/>
      </Log>
      <SendMail _="MessageJobTemplates/AnswerOnTixiError"/>
   </TixiInvalidEvent>
   <SMSInvalidPassword>
      <Log _="FailedIncomingCall"/>
      <Annotation _="SMS with invalid password received"/>
      <Sender _="&#xae;~/OA"/>
      <Time _="&#xae;~/Time"/>
      <Text _="&#xae;~/Text"/>
      </Log>
   </SMSInvalidPassword>
   <SMSInvalidEvent>
      <Log _="FailedIncomingCall"/>
      <Annotation _="SMS with invalid event received"/>
      <Sender _="&#xae;~/OA"/>
      <Time _="&#xae;~/Time"/>
      <Text _="&#xae;~/Text"/>
      </Log>
      <SendMail _="MessageJobTemplates/AnswerOnSMSError"/>
   </SMSInvalidEvent>
   <POPInvalidPassword>
```

Event handler for the Express-E-Mail error cases

Event handler for the SMS error cases

```
      <Log _="FailedIncomingCall">
      <Annotation _="POP3 email with invalid password received"/>
      <Sender _="&#xae;~/OA"/>
      <Time _="&#xae;~/Time"/>
      <Text _="&#xae;~/Text"/>
      </Log>
   </POPInvalidPassword>
   <POPInvalidEvent>
      <Log _="FailedIncomingCall">
      <Annotation _="POP3 email with invalid event received"/>
      <Sender _="&#xae;~/OA"/>
      <Time _="&#xae;~/Time"/>
      <Text _="&#xae;~/Text"/>
      </Log>
     <SendMail _="MessageJobTemplates/AnswerOnPOP3Error"/>
   </POPInvalidEvent>
 </System>
</EventHandler>
```

Event handler for the POP3 error cases

Note:
If the message(s) created by this event handler should be sent to the sender of the triggering message, the sender parameter (OA for SMS or e-mail) is only valid for messages which are sent by this specific event handler.

9.3.5.2 Message Job Templates for the answer messages

For the answer messages, separate message job templates must be created. If the address of the sender is not known at the time of configuration, the sender number can be read from the event message generated by the messages.

Database path: /TEMPLATE/MessageJobTemplate

SMS:

```
<MessageJobTemplates>
   <AnswerOnHeaterOn _="GSMSMS">
      <Recipient _="&#xae;~/OA"/>
      <Body _="Heater is On"/>
   </AnswerOnHeaterOn>
   <AnswerOnError _="GSMSMS">
      <Recipient _="&#xae;~/OA"/>
      <Body _="Command &#xae;~/Event; could not be processed"/>
   </AnswerOnError>
</MessageJobTemplates>
```

POP3:

```
<MessageJobTemplates>
   <AnswerOnHeaterOn _="SMTP">
      <Recipient _="&#xae;~/OA"/>
      <Body _="Heater is On"/>
   </AnswerOnHeaterOn>
   <AnswerOnError _="SMTP">
      <Recipient _="&#xae;~/OA"/>
      <Body _="Command &#xae;~/Event; could not be processed"/>
   </AnswerOnError>
</MessageJobTemplates>
```

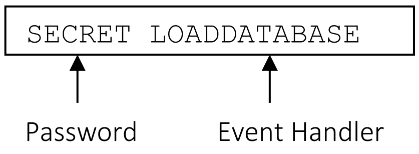
Note:
The sender address is read from the created event. In the Message Job Template, it is represented by the `&#xae;~/OA` characters (SMS, e-mail) which are a reference to the parameter provided by the event message.

9.4 Configuration via email

Additional to changing variables via incoming messages (see chapters above), it is possible to replace complete databases e.g. the AddressBook via incoming email.
The necessary event handler command "SetConfig" is explained in chapter 3.7.1.

The FP Gateway requires special message syntax to detect and process the new database content.

At first the subject line of the message has to contain the password (see chapter 9.5) and the event handler name with SetConfig command, e.g.:



The message body has to contain the databases in following syntax:

Message body syntax – edit databases	

Syntax	**`<D>`**
	   `<SetConfig _="`*`DATABASE`*`">`
	      `<Group>`
	      `Data…`
	      `</Group>`
	   `</SetConfig>`
	`</D>`

Description	Message body structure to change databases via incoming email.
Elements	***DATABASE:*** Name of database to edit, e.g. USER, ISP, PROCCFG. See chapter 14 for database names. ***Group:*** Groups inside the database, e.g. database TEMPLATE may contain groups "AddressBook", "MessageJobTemplates" and "UserTemplates"

Example

Email message body to change the location settings:

```
<D>
   <SetConfig _="USER">
      <Location>
       <CountryPrefix _="00"/>
       <CountryCode _="49"/>
       <AreaPrefix _="0"/>
       <AreaCode _="30"/>
       <LocalDialPrefix _=""/>
       <LongDialPrefix _=""/>
       <PhoneNumber _="0304019008"/>
       <InternalDialPrefix _=""/>
       <ExtensionNumber _=""/>
       <DialRules _="Tone,NoWaitForDialTone"/>
      </Location>
   </SetConfig>
</D>
```

Email message body to change the AddressBook:

```
<D>
   <SetConfig _="TEMPLATE">
      <AddressBook>
       <MySelf>
          <Email _="user@domain.com"/>
       </MySelf>
       <Receiver>
          <Email _="demo@tixi.com"/>
       </Receiver>
      </AddressBook>
   </SetConfig>
</D>
```

9.5 Authentication

To protect the message access to the device, one or more passwords must be defined. There are two ways to configure a password protection:

- Simple password.
- Sender data (CallerID, email alias) dependent.

The simple method defines a password for all senders independently of the sender device. When using the other variant, the password is valid in conjunction with a specific originating address (callerID, email alias) only. For originating address protection (callerID check) with SMS, the telephone provider and/or PBX has to support callerID presentation (CLIP). Ask your local telephone company for details.

The AccRights have to be configured in the USER database.

Database path: /USER/AccRights

Remote Control access protection

Syntax	**`<User>`**
	`   <Def_Message Plain="`*`PlainPwd`*`" Group="`*`UserGroup`*`"/>`
	`   <`*`OA_nnn`*` Plain="`*`PlainPwd`*`" Group="`*`UserGroup`*`"/>`
	`   <`*`Alias`*` Plain="`*`PlainPwd`*`" Group="`*`UserGroup`*`"/>`
	`</User>`

Description	Enables incoming message access to the FP Gateway by setting up a password protection. Either a sender-independent password can be used or one that is valid in conjunction with a specific originating address only. See chapter 3.6 for more details.
Elements	***PlainPwd:*** The password required for message access. Maximum 25 characters. ***UserGroup:*** Name of a group with service "Message" (see chapter 3.6). ***OA_nnn:*** Originating address (callerID, email address) that should be authorized to access the FP Gateway, where ***nnn*** consists of numbers only (SMS) or letters (email), thus any hyphens and other characters than numbers (or letters), must be removed. *Please note that this includes the @ sign and any dots, which must be omitted when entering e-mail addresses into this database. Accordingly, "user@example.com" becomes "userexamplecom".* ***Alias:*** Email alias name that should be authorized to access the FP Gateway. Space characters will be removed.
Example	Configure a non-sender aware password protection with the password `SECRET`: (see code below)

```
[<SetConfig _="USER" ver="y">
   <AccRights>
    <Groups>
       <MessageGroup>
        <Message AccLevel="1"/>
        </MessageGroup>
    </Groups>
    <User>
        <Def_Message Plain="SECRET"
             Group="MessageGroup"/>
    </User>
   </AccRights>
</SetConfig>]
```

Configure protection for a sender "`+49172123456789`" and the password `SECRET`:

```
[<SetConfig _="USER" ver="y">
   <AccRights>
    <Groups>
        <MessageGroup>
            <Message AccLevel="1"/>
        </MessageGroup>
    </Groups>
    <User>
        <OA_49172123456789 Plain="SECRET"
             Group="MessageGroup"/>
    </User>
   </AccRights>
</SetConfig>]
```

Configure Protection for an email sender with the alias name 'Tixi Support' and the password `SECRET`:

```
[<SetConfig _="USER" ver="y">
   <AccRights>
    <Groups>
        <MessageGroup>
            <Message AccLevel="1"/>
        </MessageGroup>
    </Groups>
    <User>
        <TixiSupport Plain="SECRET" Group="MessageGroup"/>
    </User>
   </AccRights>
</SetConfig>]
```

If alias name can not be found within user list, the email address will be checked too: Protection for an email sender without alias but with email address `tixi-support@inovolabs.com` and the password `SECRET`:

```
[<SetConfig _="USER" ver="y">
   <AccRights>
    <Groups>
        <MessageGroup>
            <Message AccLevel="1"/>
        </MessageGroup>
    </Groups>
    <User>
        <OA_Supporttixicom Plain="SECRET"
             Group="MessageGroup"/>
    </User>
   </AccRights>
</SetConfig>]
```

10. FP Gateway and PLC / Meter / Fieldbus Operation

The FP Gateway is not only to be used on its own, but even in conjunction with PLC devices, meters and other kinds of controllers. As the FP Gateway got most common protocols already implemented, connecting it to a PLC or meter is very simple.

Once the connection is established physically via RS232, RS422/485, MPI, M-Bus or LAN interface, the FP Gateway system properties may be enhanced by any variable of the external device. The FP Gateway will then be able to read the variable values, use them to trigger events, log the values and send the logfiles. Based on logical instructions or incoming messages, the FP Gateway may even write into devices.

There's a variety of systems to be supported by the FP Gateway, which got the respective protocols already implemented. These PLCs or meters may be connected to the FP Gateway without any change in programming or configuration, just by means of their communication interface:

- Mitsubishi ALPHA2, MELSEC FX (FX1, FX2, FX3U, FU)
- Siemens Simatic S7-200, S7-300/400 (MPI), S7-1200, S7-1500 (ISO-ON-TCP)
- Siemens Logo! 8.X (via Modbus)
- VIPA (100V, 200V, 300V via GreenCable)
- Moeller / Eaton Easy 400/500/600/700/800/MFD, Easy Control, XC/XVC, PS4
- Moeller / Eaton Easy E4 (via Modbus)
- SAIA S-Bus (PCD2, PCS)
- Carel Macroplus
- Modbus RTU, Modbus TCP (Master & Slave)
- Crouzet Millenium 2 + 3
- Schneider Electric Zelio, Twido, MODICON
- ABB CL, AC010, AC31, AC500
- Tixibus
- M-Bus
- ABB inverters (Aurora protocol)

Detailed information on configuration and usage of FP IoT Gateways along with PLCs and other devices can be found within the PLC Manual, which is available on our website.

11. File transfer protocols FTP / SFTP

The FP Gateway implements FTP and SFTP clients which allows you to transfer files to file servers using the standardized file transfer protocols FTP and SFTP.

11.1 FTP Client

The FP gateway implements the command `FTPPut` which can be used within an event handler to send files to any standard FTP server.

EventHandler definition

```
<EventHandler>
  <Alarm_0>
    <FTPPut _="MessageJobTemplates/Alarm_0">
       <OnOK    _="OnOKEvent"/>       (optional)
       <OnError _="OnErrorEvent"/>    (optional)
    </FTPPut>
  </Alarm_0>
</EventHandler>
```

The ***FTPPut*** command uses a MessageJobTemplate (*Alarm_0* in this example) which defines the protocol, recipient, sender, file content (body) and the FTP file name.

The optional event handlers *OnOKEvent* and *OnErrorEvent* will be triggered depending on the result of the file transfer operation. *OnOKEvent* is called if the FTP file was transferred successfully to the FTP server. If the file transfer failed the event handler *OnErrorEvent* will be called.

MessageJobTemplate definition

```
<MessageJobTemplates>
  <Alarm_0 _="FTP">
     <Recipient _="/D/AddressBook/Contact_0"/>
     <Sender _="/D/AddressBook/MySelf"/>
     <Body _="/D/UserTemplates/FTP_Text"/>
     <FTPFileName _=
       "internal/users/Joe/Data-&#xae;/TIMES/YYYY_MM_DD;_&#xae;/Times/HH_MM_SS;"/>
  </Alarm_0>
</MessageJobTemplates>
```

The protocol is specified by the entry "**FTP**". *Alarm_0* is the name of the event handler.

The FTP ***Recipient*** is defined in the AddressBook and contains the FTP user name (**FTPUser**), FTP password (**FTPPassword**), FTP server address (**FTPServer**), FTP port (**FTPPort**; optional; default = 21).

The ***Sender*** links to the address book entry "`MySelf`". The ***Body*** defines the content of the FTP file and is linked to a **UserTemplate** (*FTP_Text* in the example above).

The ***FTPFileName*** configures the file name of the file created on the FTP server. It may contain a path, separated by "/". It may also contain references like the serial number of the device or a timestamp. Please note that it is not allowed to use system properties which contain "/" because this would be interpreted as a path. So for example the system properties `/TIMES/Data` or `/TIMES/Time` may not be used !

AddressBook definition

```
<AddressBook>
  <MySelf>
     <Email _="user@domain.com"/>
     <SMS_No _="+49-30-1234567"/>
  </MySelf>

  <Contact_0>
    <FTPUser _="anonymous"/>
    <FTPPassword _="user@domain.com"/>
    <FTPServer _="ftp.domain.com"/>
    <FTPPort _="21"/>      (optional, default=21)
  </Contact_0>
</AddressBook>
```

`FTPUser`	FTP username
`FTPPassword`	FTP password
`FTPServer`	FTP server address (domain name or IP address)
`FTPPort`	(optional) FTP port (default: 21). (every address book entry may have a different FTP port definition)

UserTemplate definition

The UserTemplate defines the content of the FTP file. The file may contain log data as well as fixed text data and variables.

```
<UserTemplates>
  <FTP_Text>
    <IncludeLog _="Datalogging_0" range="#10-"/>
    <IncludeLogTXT _="Datalogging_1" flags="NoId" type="CSV"
             colsep="," range="#10-"/>
  </FTP_Text>
</UserTemplates>
```

See chapter 3.8 for detailed info how to include text and variables into the FTP file.

Possible log data formats are XML (***IncludeLog***) and CSV (***IncludeLogTXT***). See chapter 4.6 for the complete range of possible options.

Service Routing

It is possible to specify the interface which should be used to send out the FTP file. This is defined in the `/ISP/ISP` database, section `OUT`:

```
<OUT>
   <FTPPut _="ServicePath"/>
</OUT>
```

The following values are valid fort he *`ServicePath`* entry (case sensitive !):

`MODEM`	mobile modem (default)
`Ethernet`	LAN connection
`VPN`	VPN tunnel

11.2 SFTP Client

The FP gateway implements an SFTP client which can be used within an event handler to send files to any standard SFTP server using the **S**SH **F**ile **T**ransfer **P**rotocol (also known as **S**ecure **F**ile **T**ransfer **P**rotocol). The event handler command is `SFTPPut`.

EventHandler definition

```
<EventHandler>
  <Alarm_0>
    <SFTPPut  _="MessageJobTemplates/Alarm_0">
       <OnOK    _="OnOKEvent"/>      (optional)
       <OnError _="OnErrorEvent"/>   (optional)
    </SFTPPut>
  </Alarm_0>
</EventHandler>
```

The ***SFTPPut*** command uses a MessageJobTemplate (*Alarm_0* in this example) which defines the protocol, recipient, sender, file content (body) and the FTP file name.

The optional event handlers *OnOKEvent* and *OnErrorEvent* will be triggered depending on the result of the file transfer operation. *OnOKEvent* is called if the FTP file was transferred successfully to the SFTP server. If the file transfer failed the event handler *OnErrorEvent* will be called.

MessageJobTemplate definition

```
<MessageJobTemplates>
  <Alarm_0 _="SFTP">
     <Recipient _="/D/AddressBook/Contact_0"/>
     <Sender _="/D/AddressBook/MySelf"/>
     <Body _="/D/UserTemplates/FTP_Text"/>
     <SFTPFileName _=
       "internal/users/Joe/Data-&#xae;/TIMES/YYYY_MM_DD;_&#xae;/Times/HH_MM_SS;"/>
  </Alarm_0>
</MessageJobTemplates>
```

The protocol is specified by the entry "**SFTP**". *Alarm_0* is the name of the event handler.

The SFTP ***Recipient*** is defined in the AddressBook and contains the SFTP user name (**SFTPUser**), SFTP password (**SFTPPassword**), FTP server address (**SFTPServer**), SFTP port (**SFTPPort**; optional; default = 22).

The ***Sender*** links to the address book entry "`MySelf`". The ***Body*** defines the content of the FTP file and is linked to a **UserTemplate** (*FTP_Text* in the example above).

The ***SFTPFileName*** configures the file name of the file created on the SFTP server. It may contain a path, separated by "/". It may also contain references like the serial number of the device or a timestamp. Please note that it is not allowed to use system properties which contain "/" because this would be interpreted as a path. So for example the system properties `/TIMES/Data` or `/TIMES/Time` may not be used !

AddressBook definition

```
<AddressBook>
  <MySelf>
     <Email  _="user@domain.com"/>
     <SMS_No  _="+49-30-1234567"/>
  </MySelf>

  <Contact_0>
    <SFTPUser _="anonymous"/>
    <SFTPPassword _="user@domain.com"/>
    <SFTPServer _="ftp.domain.com"/>
    <SFTPPort _="22"/>   (optional, default=22)
  </Contact_0>
</AddressBook>
```

`SFTPUser`	SFTP username
`SFTPPassword`	SFTP password
`SFTPServer`	SFTP server address (domain name or IP address)
`SFTPPort`	(optional) SFTP port (default: 22). (every address book entry may have a different SFTP port definition)

UserTemplate definition

The **UserTemplate** defines the content of the SFTP file. The file may contain log data as well as fixed text data and variables.

```
<UserTemplates>
  <FTP_Text>
    <IncludeLog _="Datalogging_0" range="#10-"/>
    <IncludeLogTXT _="Datalogging_1" flags="NoId" type="CSV"
              colsep="," range="#10-"/>
  </FTP_Text>
</UserTemplates>
```

See chapter 3.8 for detailed info how to include text and variables into the SFTP file.

Possible log data formats are XML (***IncludeLog***) and CSV (***IncludeLogTXT***). See chapter 4.6 for the complete range of possible options.

Service Routing

It is possible to specify the interface which should be used to send out the SFTP file. This is defined in the `/ISP/ISP` database, section `OUT`:

```
<OUT>
   <SFTPPut _="ServicePath"/>
</OUT>
```

The following values are valid fort he *`ServicePath`* entry (case sensitive !):

`MODEM`	mobile modem (default)
`Ethernet`	LAN connection
`VPN`	VPN tunnel

12. Service-Routing

A service is one of the following features:

- Email sending (`SMTP`)
- Email receiving (`POP3`)
- Remote connections (`CBIS`)
- Calling an external URL (`URLSend`)
- Getting internet time (`INetTime` = DAYTIME / NTP)
- Avantgarde / Alnamic cloud connector (`HTTPConn`)
- Cloud connections (`CloudConn`)
  valid for: Universal MQTT, Juconn MQTT, Telekom CoT, Cumulocity cloud
- FTP connections (`FTPPut`)
- SFTP connections (`SFTPPut`)
- VPN tunnel (`VPN`)

All these services are using the default WAN connection (Ethernet or modem connection).

Via the /ISP/ISP/OUT database it is possible to specify the WAN interface which should be used for each of the above services.

```
[<SetConfig _="ISP/ISP" ver="y">
    <!-- define communication interface for services -->
    <OUT>
        <SMTP      _="ServicePath"/>
        <CBIS      _="ServicePath"/>
        <POP3      _="ServicePath"/>
        <URLSend   _="ServicePath"/>
        <INetTime  _="ServicePath"/>
        <HTTPConn  _="ServicePath"/>
        <CloudConn _="ServicePath"/>
        <FTPPut    _="ServicePath"/>
        <SFTPPut   _="ServicePath"/>
        <VPN       _="ServicePath"/>
    </OUT>
</SetConfig>]
```

The following values are valid fort he *`ServicePath`* entry (case sensitive !):

`MODEM`	mobile modem connection (default)
`Ethernet`	LAN connection
`VPN`	VPN tunnel

13. Addresses of serial interfaces and I/Os

FP Gateways					
Device	FP Gateway				
Description	GSM	GSM	GSM	GSM	V.90 / GSM / ISDN
Product Code	HG121 / HG421	HG127 / HG427	HG141 / HG441	HG147 / HG447	HG425-2S0
Serial interfaces	1xRS232-F, 1xRS232-M	1xRS232-F, 1xRS232-M	1xRS232-F, 1xRS422/485	1xRS232-F, 1xRS422/485	1xRS232-F, 1xRS232-M
I/Os	-	2/3 + 10V AI	-	2/3 + 10V AI	2/1 + 10V AI + 2 S0
Systempath:					
RS232-1	COM1	COM1	COM1	COM1	COM1
RS232-2	COM2	-	-	-	COM2
RS422/485	-	COM2	COM2	COM2	-
Inputs	-	/Process/MB/IO/I/Px	-	/Process/MB/IO/I/Px	/Process/MB/IO/I/Px
Outputs	-	/Process/MB/IO/Q/Px	-	/Process/MB/IO/Q/Px	/Process/MB/IO/Q/Px
Analog input	-	/Process/MB/A/AI/P0	-	/Process/MB/A/AI/P0	/Process/MB/A/AI/P0
S0-interface	-	-	-	-	/Process/I3e/AI/Px

FP Gateways				
Device	FP Gateway			
Description	GSM	GSM	GSM	GSM
Product Code	HG423-M25/60/100	HG443-M25/60/100	HG171 / HG471	HG176 / HG476
Serial interfaces	1xRS232-F, 1xRS232-M	1xRS232-F, 1xRS422/485	1xRS232-F, 1xMPI	1xRS232-F, 1xMPI
I/Os	2/1 + 10V AI	2/1 + 10V AI	-	2/2 + 10V AI
Systempath:				
RS232-1	COM1	COM1	COM1	COM1
RS232-2	COM2	-	COM2	COM2
RS422/485	-	COM2	-	-
Inputs	COM3	COM3		
Outputs	/Process/MB/IO/I/Px	/Process/MB/IO/I/Px	-	/Process/MB/IO/I/Px
Analog input	/Process/MB/IO/Q/Px	/Process/MB/IO/Q/Px	-	/Process/MB/IO/Q/Px
S0-interface	/Process/MB/A/AI/P0	/Process/MB/A/AI/P0	-	/Process/MB/A/AI/P0

FP Gateways					
Device	FP Gateway				
Description	LAN/WLAN	LAN/WLAN	LAN/WLAN	LAN/WLAN	LAN/WLAN
Product Code	HE121 / HE421 / HW 121 / HW421	HE127 / HE427 / HW127 / HW427	HE141 / HE441 / HW141 / HW441	HE147 / HE447 / HW147 / HW447	HE425-2S0, HW425-2S0

Serial interfaces	1xRS232-F, 1xRS232-M	1xRS232-F, 1xRS232-M	1xRS232-F, 1xRS422/485	1xRS232-F, 1xRS422/485	1xRS232-F, 1xRS232-M
I/Os	-	2/3 + 10V AI	-	2/3 + 10V AI	2/1 + 10V AI + 2 S0
Systempath:					
RS232-1	COM1	COM1	COM1	COM1	COM1
RS232-2	COM2	-	-	-	COM2
RS422/485	-	COM2	COM2	COM2	-
Inputs	-	/Process/MB/IO/I/Px	-	/Process/MB/IO/I/Px	/Process/MB/IO/I/Px
Outputs	-	/Process/MB/IO/Q/Px	-	/Process/MB/IO/Q/Px	/Process/MB/IO/Q/Px
Analog input	-	/Process/MB/A/AI/P0	-	/Process/MB/A/AI/P0	/Process/MB/A/AI/P0
S0-interface	-	-	-	-	/Process/I3e/AI/Px

FP Gateways				
Device	FP Gateway			
Description	LAN/WLAN	LAN/WLAN	LAN/WLAN	LAN/WLAN
Product Code	HE423-M25/60/100 / HW423-M25/60/100	HE443-M25/60/100 / HW443-M25/60/100	HE171 / HE471 / HW171 / HW471	HE176 / HE476 / HW176 / HW476
Serial interfaces	1xRS232-F, 1xRS232-M	1xRS232-F, 1xRS422/485	1xRS232-F, 1xMPI	1xRS232-F, 1xMPI
I/Os	2/1 + 10V AI	2/1 + 10V AI	-	2/2 + 10V AI
Systempath:				
RS232-1	COM1	COM1	COM1	COM1
RS232-2	COM2	-	COM2	COM2
RS422/485	-	COM2	-	-
Inputs	COM3	COM3		
Outputs	/Process/MB/IO/I/Px	/Process/MB/IO/I/Px	-	/Process/MB/IO/I/Px
Analog input	/Process/MB/IO/Q/Px	/Process/MB/IO/Q/Px	-	/Process/MB/IO/Q/Px
S0-interface	/Process/MB/A/AI/P0	/Process/MB/A/AI/P0	-	/Process/MB/A/AI/P0

IO-Extensions			
Device Description	FP Gateway IO-Extension, FP Gateway IO-Extension		
Product Code	XP84D	XP84DR	XP88AD
I/Os	8/4	8/4	8/0 + 8x10V AI
Systempath:			
Cx addresses	C40, C42, C44, C46, C48, C4A, C4C, C4E	C40, C42, C44, C46, C48, C4A, C4C, C4E	C40, C42, C44, C46, C48, C4A, C4C, C4E
Inputs	/Process/C4x/I/Px	/Process/C4x/I/Px	/Process/C4x/I/Px

Outputs	/Process/C4x/Q/Px	/Process/C4x/Q/Px	-
Analog input	-	-	/Process/C4x/AI/P0

13.1 Bit / Byte / Word / Dword addressing of I/Os

The bit values of the FP Gateway I/Os can also be addresses as bytes, words and dwords.

Bit / Byte / Word / Dword Addressing

Syntax ***/Process/[ModuleAddress]/[PortGroup]/[PortType][Range]/P[PortIndex]***

Description Addressing the input and output ports of a FP Gateway. Port bits can be addressed in different ways. It is possible to address a single port or sequences of ports. Assuming a 32-bit port, the bits can be addressed in the following ways:

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Single Bit Access:example: /Process/MB/I/IO/P4

00	01	02	03	▨ 04	05	06	07	08	09	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Byte Access: example: /Process/MB/IO/IB/P2

0	1	▨ 2	3

Word Access: example: /Process/MB/IO/IW/P1

0	▨ 1

Double Word Access: example /Process/MB/IO/ID/P0

▨ 0

Elements The port is addressed by a slash divided path notation:

ModuleAddress:

MB	FP Gateway mainboard
C40...C4E	HEX number corresponds to the setting of the address jumper of the extension modules.

PortGroup (only on ModuleAddress "MB"):

IO	Group of digital in- and outputs

PortType:

PortType	DataType	Value Range
I	Input Bitfield	0,1
IB	Input Byte	0..255
IW	Input Word	0...65.535
ID	Input DWord	0...4.294.967.295
Q	Output Bitfield	0,1
QB	Output Byte	0..255
QW	Output Word	0...65.535
QD	Output DWord	0...4.294.967.295

PortIndex:

0...*n*	Zero based index number of the item in the port.

Example 5. bit of the input bit field of the module with the address 40.

```
/Process/C40/I/P4
```

14. System Properties

This appendix lists the available system properties of the FP Gateway which can be read or written by the Get and Set commands. The tables contain the single system properties. The complete path to address a system property must be combined by the headline of the table and the name of the system property separated by a slash character.

Example
To address the serial number simply write:
/SerialNo (table has no headline).

To address the version number of the firmware write:

```
/OEM/Firmware/Version
```

Headline

Name from table

It's possible to include the system properties into message text, e.g.:

```
<L _="Firmware-Version: &#xae;/OEM/Firmware/Version"/>
```

In each table you can find 5 device types for example with an ‚x' if they have the specific System Property.

/BoxMode

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
						String	yes	Current system mode (HM and HG only) Modem TiXML

/Components

Name	WG 660	HE 652	WG 640	HU 643	HU 651	Type	Read only	Description
	x	x	x	x	x	String	yes	List of components of the device, for example: RTC Modem0 FlashOnboard C8

/EEPROM

Name	WG 660	HE 652	WG 640	HU 643	HU 651	Type	Read only	Description
GM						String	yes	Interface address of GSM modem, copied from configuration
Pin1	x					String	yes	PIN1 for GSM SIM card, copied from configuration
Pin2						String	yes	PIN2 for GSM SIM card, copied from configuration

LED0	x	x	x	x	X	Int	no	State of the signal LED
BId			x		x	String	yes	Build ID; ID of the config.txt of the device

/Ethernet

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
AssignedIP	x	x	x	x	x	String	yes	IP address assigned via DHCP or configuration
SubnetMask	x	x	x	x	x	String	yes	SubnetMask assigned via DHCP or configuration
Gateway	x	x	x	x	x	String	yes	Gateway assigned via DHCP or configuration
DNS_1	x	x	x	x	x	String	yes	First DNS server assigned via DHCP or configuration
DNS_2						String	yes	Second DNS server assigned via DHCP or configuration
Link	x	x	x	x	x	Uint16	yes	Active link speed of ethernet interface. "0", "10" or "100" (MBit/s)
LinkState	x	x	x	x	x	Bit	yes	Status of ethernet link. 0=disconnected, 1=connected
MAC	x	x	x	x	x	String	yes	Network interface MAC address

/EVENT

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description

/FeatureList

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
	x	x	x	x	x	String	yes	List of the features (services) of the product, for example: Modem Mode Send Remote Modem-Mode Script Send Job Result Processor

/Firmware

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
Version	x	x	x	x	x	String	yes	Version number of the firmware, for example: 3.06.60.000 (G5)/5.2.020 (G6)
Date	x	x	x	x	x	String	yes	Date of build

/FreeFileSize

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
	x	x	x	x	x	Uint32	yes	Free memory in file system (Bytes)

/FreeRAMSize

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
	x	x	x	x	x	Uint32	yes	Free memory in RAM

/GPRSLinkState

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
	x		x	x	x	Uint16	yes	The GPRSLinkState is a variable, which displays the present situation of the GPRS connection. (*related to LocalIPAddr, RemoteIPAddr) Values: 0 The system is booted and a GPRS connection was not built up ever before or after sending the TiXML command GPRSDisconnect. 1 A GPRS connection exists.

/GSM

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
Reg	X		x			Uint16	yes	Shows network registration state
Reg_Text	X		x			String	yes	Verbose network registration state
Operator	X		x			String	yes	Name of active GSM operator
Quality	X		x			Uint16	yes	GSM signal strength, see ETSI GSM 05.08 0: -113dBm or less 1: -111 dBm 2-30: -109 to -53 dBm 32: -51dBm or greater 99: not known or not detectable
BitErrorRate	X		x			Uint16	yes	as RXQUAL values, see ETSI GSM 05.08
Account						Uint16	yes	Credit left on SIM card (Germany: €€cc)

DaysLeft						Uint16	yes	Days until SIM card expiration
State			x	x	x	String	yes	GSM error state, e.g. "missing SIM Card"

/GSM/FD

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
	x	x	x	x	x			GSM SIM provider phonebook entries in syntax `<Contact_name _="phone_number"/>.` Special characters in contact names will be replaced by underscore.

/GSM/SM

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
	x	x	x	x	x			GSM SIM phonebook entries in syntax `<Contact_name _="phone_number"/>.` Special characters in contact names will be replaced by underscore.

/HardwareID

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
	x	x	x	x	x	String	yes	Device hardware code

/Hardware/Modules

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
RTC	x	x	x	x	x	String	yes	RTC type, not defined if no RTC is present. for example: RTC4513
Jumper						String	yes	Jumper to prevent firmware upload, not defined if not present.
Modem0	x	x	x	x	x	String	yes	Type of Modem #0, not defined if not present. for example: HG421
Modem1						String	yes	Type of Modem #1, not defined if not present.
GsmModule	x		x	x	x	String	yes	Type of GSM modem, if present
FlashOnboard	x	x	x	x	x	String	yes	Flag indicating whether flash is on the main board or not,

								undefined if no flash is present, 'x' else.
FlashExtension						String	yes	Flag indicating whether a flash extension board is present, undefined if no flash extension board is present, 'x' else.
PowerSupply						String	yes	Hardware version of the FP Gateway (1 or 2 Ampere) for example: 2.0A
COMx	x	x	x	x	x	String	yes	Interface type of COMx
MBAIO						String	yes	Type of analog I/O module
MBDIO						String	yes	Type of digital I/O module
Ethernet						String	yes	Network controller chip type
Cxxx	x		x			String	yes	extension module (e.g. C42,C44,C44,C46,C48,C4a,C4c ,C4e, C010, C012, C032, C094, C696 etc.)
ModuleVer	x		x	x	x	String	yes	
IMEI	x		x	x	x	String	yes	
IMSI	x		x			String	yes	The **I**nternational **M**obile **S**ubscriber **I**dentity serves to identify network users clearly in GSM- and UMTS-mobile networks (internal terminal identification). Together with other data, the IMSI is saved at a SIM (Subscriber Identity Module) card. The IMSI-number is worldwide unique for each customer. (The IMSI has nothing to do with the telephone number which is assigned to the SIM card.)
MSISDN	x					String	yes	Displayed, when the used GSM module sends the own telephone number on the command AT+CNUM (number has to be deposited and saved on the SIM card).
ETH1	x	x	x		X	String	yes	
MainBoardDigital	x	x	x	x	x	String	yes	Type of Mainboard

/Hardware/RAM

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
Attributes	x	x	x	x	x	String	yes	Code describing the attributes of the system RAM, not defined if no information is present, for example: 14680064
Type						String	yes	Code describing the Type of the system RAM,

								not defined if no information is present.
Size	x	x	x	x	x	String	yes	Detected size of the system RAM in bytes, not defined if no information is present, for example: 524288-> 512 KByte

/Hardware/ROM

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
Attributes	x	x	x	x	x	String	yes	Code describing the attributes of the system ROM, not defined if no information is present. for example: 0
Type						String	yes	Code describing the type of the system ROM, not defined if no information is present.
Size	x	x	x	x	x	String	yes	Detected size of the system ROM in bytes, not defined if no information is present.

/Hardware/FileSystem

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
Attributes	x	x	x	x	x	String	yes	Code describing the attributes of the system RAM, not defined if no information is present, for example: 0
Type	x	x	x	x	x	String	yes	Code describing the type of the system RAM, not defined if no information is present.
Size	x	x	x	x	x	String	yes	Detected size of the system RAM in bytes, not defined if no information is present, for example: 262144 -> 2 MByte

/LicenseRef

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
HW_Rev	x	x	x	x	x	String	yes	Hardware Revision B-Board
Oem	x	x	x	x	x	String	yes	
OName	x	x	x	x	x	String	yes	
PClass	x	x	x	x	x	String	yes	
ProdName	x	x	x	x	x	String	yes	
LicenseID	x	x	x	x	x	String	yes	

ProductID	x	x	x	x	x	String	yes	
UDID	x	x	x	x	x	String	yes	

/Linux

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
SystemInfo	x	x	x	x	x	String	yes	Installed Linux kernel with date
Uboot	x	x	x	x	x	String	yes	Date uBoot

/LocalIPAddr

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
	x		x			String	yes	IP address of the modem (assigned during PPP connection, *see RemoteIPAddr, GPRSLinkState)

/LogCounter

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
Logfilename						Uint16	no	Number of entries in the specified Logfile. Value can be changed by "set" command, new entries will be added.
Event	x		x	x	x			
JobReport	x		x	x	x			
Login	x		x	x	x			
SupportLog	x	x	x	x	x			
FatalSystemError	x	x	x	x	x			
IncomingMessage			x	x	x			
FailedIncomingCall			x	x	x			

/PNP_String

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
	x	x	x	x	x	String	yes	Plug and Play string of the device, for example: TIX2027\02582501\MODEM\AMB3100\Tixi Data Gateway LAN

/Process/CloudConn

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
ConnectionType	x			x	x	String	yes	http(s), mqtt, …
ConnectionState	x			x	x	String	yes	0 not connected 1 connected
ConnectionStateMsg	x			x	x	String	yes	Connected or not connected
Tenant	x			x	x	String	yes	With which instance it is connected
User	x			x	x	String	yes	Name under which it is created

/Process/COMxPollActive

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
	x		x	x	x	Uint16	yes	0 no Bus at the Comport x active 1 active Bus on Comport x

/Process/ETHPollActive

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
	x		x	x	x	Uint16	yes	0 no Bus active 1 Bus active (data is pulled over the interface)

/Process/HTTPConn

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
	x	x	x	x	x	String	yes	Data for the Avantgarde-connector

/Process/IBMConn

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
	x	x	x	x	x	String	yes	Data for the IBM-connector

/Process/Program

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
Mode	x	x	x	x	x	String	no	Processing Mode of the program: Run Processing is running Stop Processing is

								stopped.

/Process/SerialIP

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
	x					String	yes	Comport over which the serial IP is running (1-5), 0 = Ethernet

/Process/SysIO

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
P0			x			Uint16	yes	Power supply 1 Power supply exists 0 No power supply (battery mode)
P1			x			Uint16	yes	1-Wire overload 1 1-Wire overload (short circuit, to many sensors) 0 No overload
P2			x			Uint16	yes	M-Bus overload 1 M-Bus overload (short circuit, to many counters) 0 No overload
P3			x			Uint16	yes	1 Akku is being charged 0 Charge process completed
P4			x			Uint16	yes	Battery voltage in millivolt

/Process/VPN

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
ConnectionState	x		x	x	x	String	yes	0 not connected 1 connected
ConnectionSateMsg			x			String	yes	Not connected / connecting / connected
Inteface			x			String	yes	VPN tunnel over modem or ethernet
IP			x			String	yes	IP in the VPN tunnel

/Process/MB

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
FirstCycle	x	x	x	x	x	Uint16	no	Predefined process variable (used to detect power on situation): 1 The first processing cycle runs.

								0 The first processing cycle is done.
PollButton	x	x	x	x	x	Uint16	yes	Status of the Service Button. 1=pressed, 0=not pressed
SignalLED	x	x	x	x	x	Uint16	no	Status of the "Signal" LED. Range 0 – 27 (see chapter 6.8)
ModemOffHook	x		x	x	x	Uint16	yes	State of the modem 0: On hook 1: Off hook (outgoing call) 2: Off hook (incoming call)
TransMode						Uint16	yes	TransMode state: 0: no TransMode established 1: TransMode to COM1 established 2: TransMode to COM2 established
MaxCycleTime	x	x	x	x	x	Uint16	yes	Longest cycle time
CycleTime	x	x	x	x	x	Uint16	yes	Current cycle time
Mount	x	x	x	x	x			Shows if a USB memory stick or SD card is mounted.
CPULoad	x	x	x	x	x			Shows the processor load in %
TixmlQueue	x		x	x	x			Number of commands, that are currently in the queue.

/Process/MB/IO

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
I	x	x	x	x	x			Input
IB	x	x	x	x	x			Input
IW	x	x	x	x	x			Input
ID	x	x	x	x	x			Input
Q	x		x	x				Output
QB	x		x	x				Output
QW	x		x	x				Output
QD	x		x	x				Output

/Process/Cxxx

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description

/RemoteIPAddr

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
	x		x			String	yes	IP of the GPRS gateway

/SerialNo

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
	x	x	x	x	x	String	yes	Serial Number of the device, for example: 00238801

/TIMES

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
TIME	x	x	x	x	x	String	yes	Current system time of the day:hour:minutes:seconds
DATE	x	x	x	x	x	String	yes	Current system date year/month/day
RFC822Date	x	x	x	x	x	String	yes	Current system date and time in the email format according to RFC 822, for example: Fri,13 Jul 01 13:56:00 +0100
PowerOffTime	x	x	x	x	x	String	yes	System time of the last power off event with (resolution of 1 minute.) year/month/day/hours/minutes/seconds
PowerOnTime	x	x	x	x	x	String	yes	System time of the last power on event with year/month/day/hours/minutes/seconds
DAYOFWEEK	x	x	x	x	x	String	yes	Sunday, Monday, …Saturday
DAYOFWEEKNO	x	x	x	x	x	String	yes	Day of week: 0-6
YYYY_MM_DD	x	x	x	x	x	String	yes	Current system date year_month_day.
HH_MM_SS	x	x	x	x	x	String	yes	Current system time of the day hour_minutes_seconds.
HEXDATE	x	x	x	x	x	String	yes	Current system time in seconds since 1.1.1970 as hex value.
ISO8601DATE	x	x	x	x	x			Current date and time of day according tot he time zone that is set (Z = time zone UTC)
YYMM	x	x	x	x	x			

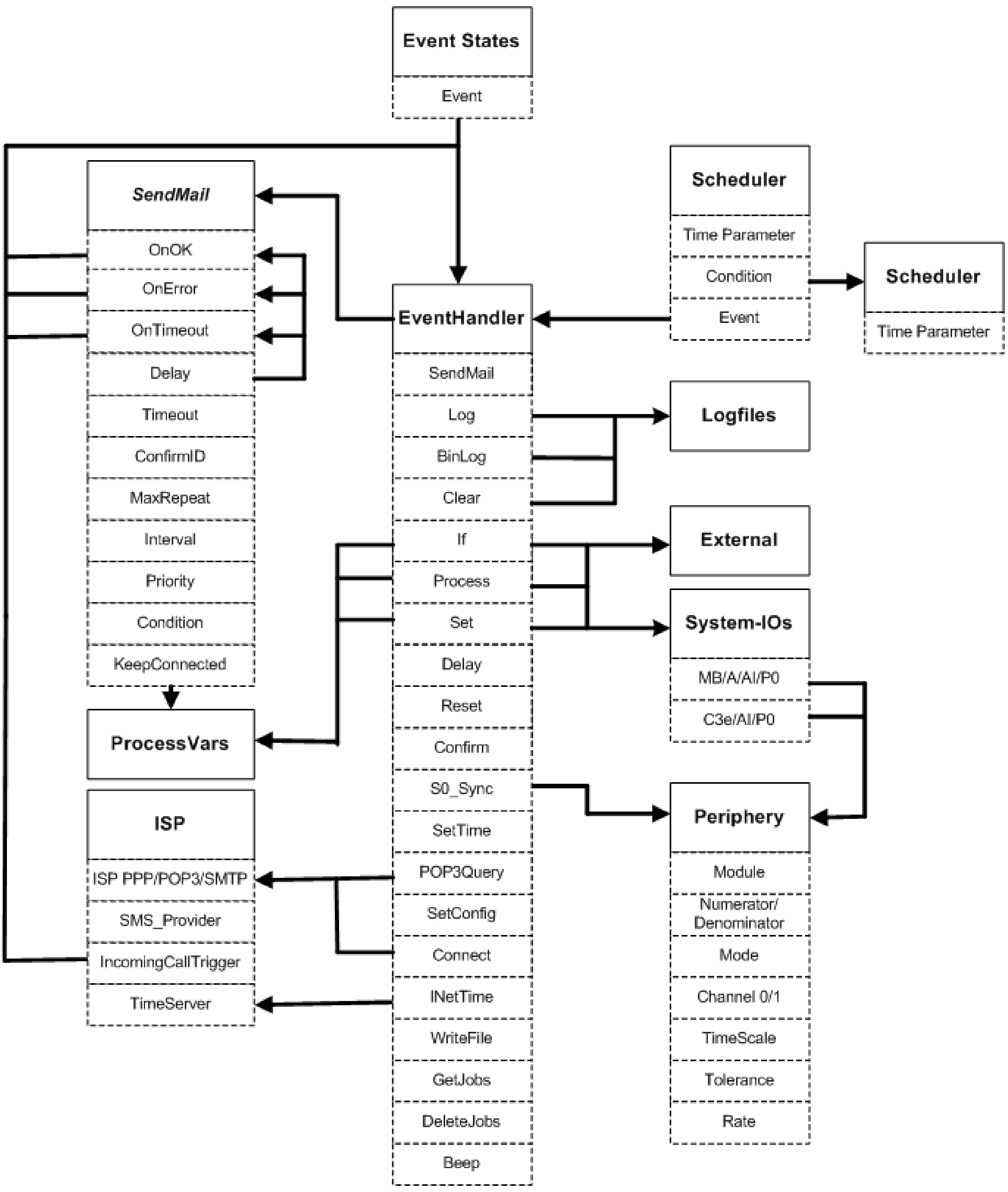
/WLAN

Name	WG 660	HE 651	WG 640	HU 643	HU 651	Type	Read only	Description
AssignedIP	x	x			x	String	yes	IP address assigned via DHCP or configuration
SubnetMask	x	x			x	String	yes	SubnetMask assigned via DHCP or configuration
Gateway		x			x	String	yes	Gateway assigned via DHCP or configuration

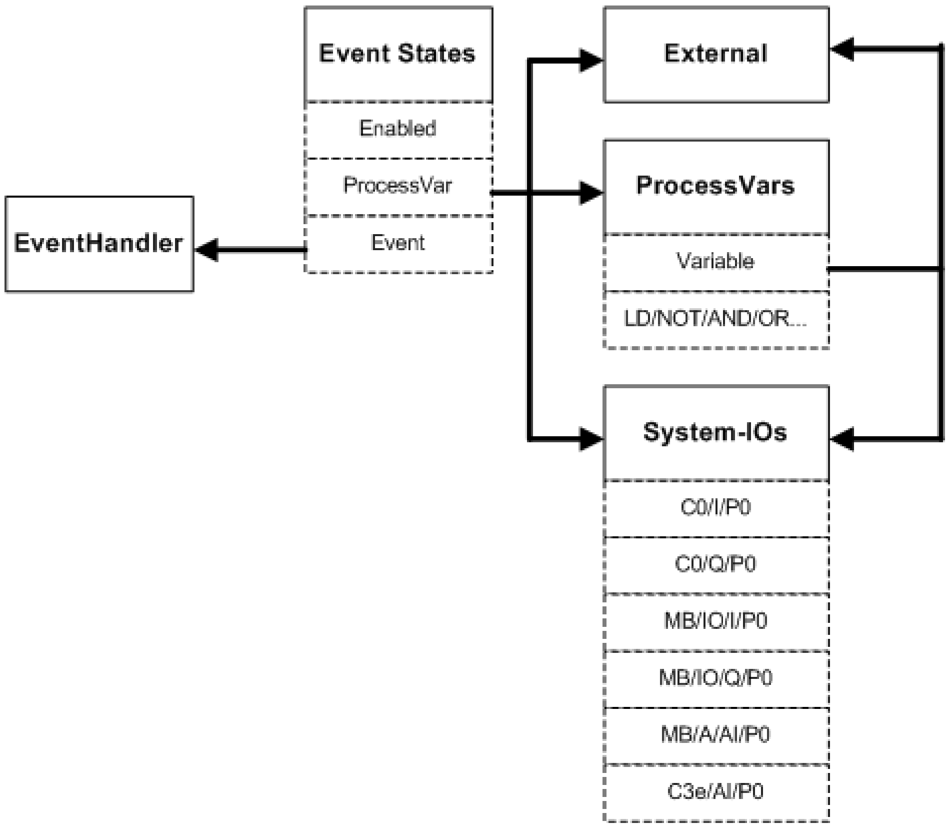
DNS_1	x	x			X	String	yes	First DNS server assigned via DHCP or configuration
DNS_2						String	yes	Second DNS server assigned via DHCP or configuration
Rate	x	x				String	yes	Active link speed of WLAN interface, e.g."54 Mbps"
Link						Uint16	yes	Active link speed of WLAN interface (MBit/s).
LinkState						Uint16	yes	Status of WLAN link. 0=disconnected, 1=connected
TxPower						String	yes	WLAN transmission power
MAC	x	x			x	String	yes	WLAN interface MAC address
SSID	x	x			x	String	yes	Service Set Identifier of connected access point
BSSID						String	yes	Basic Service Set Identifier of connected access point
SNR	x	x				String	yes	WLAN signal to noise ratio, e.g. "48/96"
Reason						String	yes	WLAN error reason
RSN	x	x			x			
Role	x	x			x			
HostName	x	x			x			
Channel					x			

15. Project structure and connections

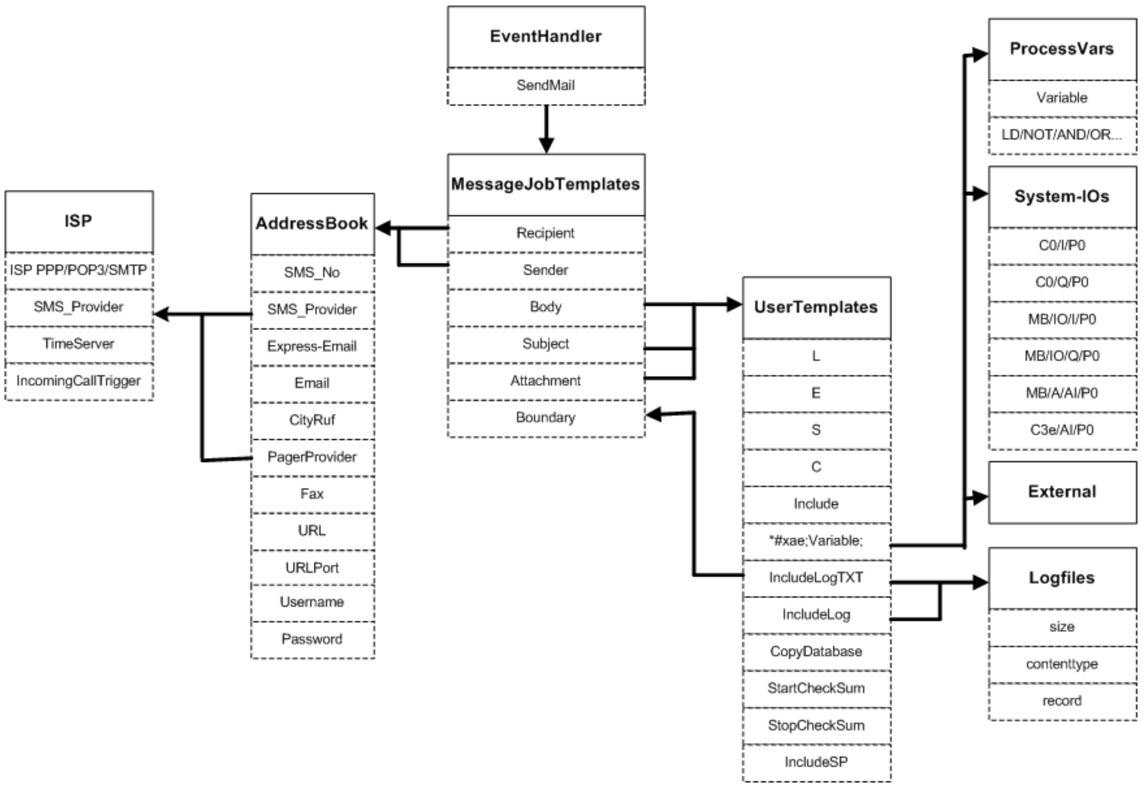
15.1 Event Handler, Scheduler



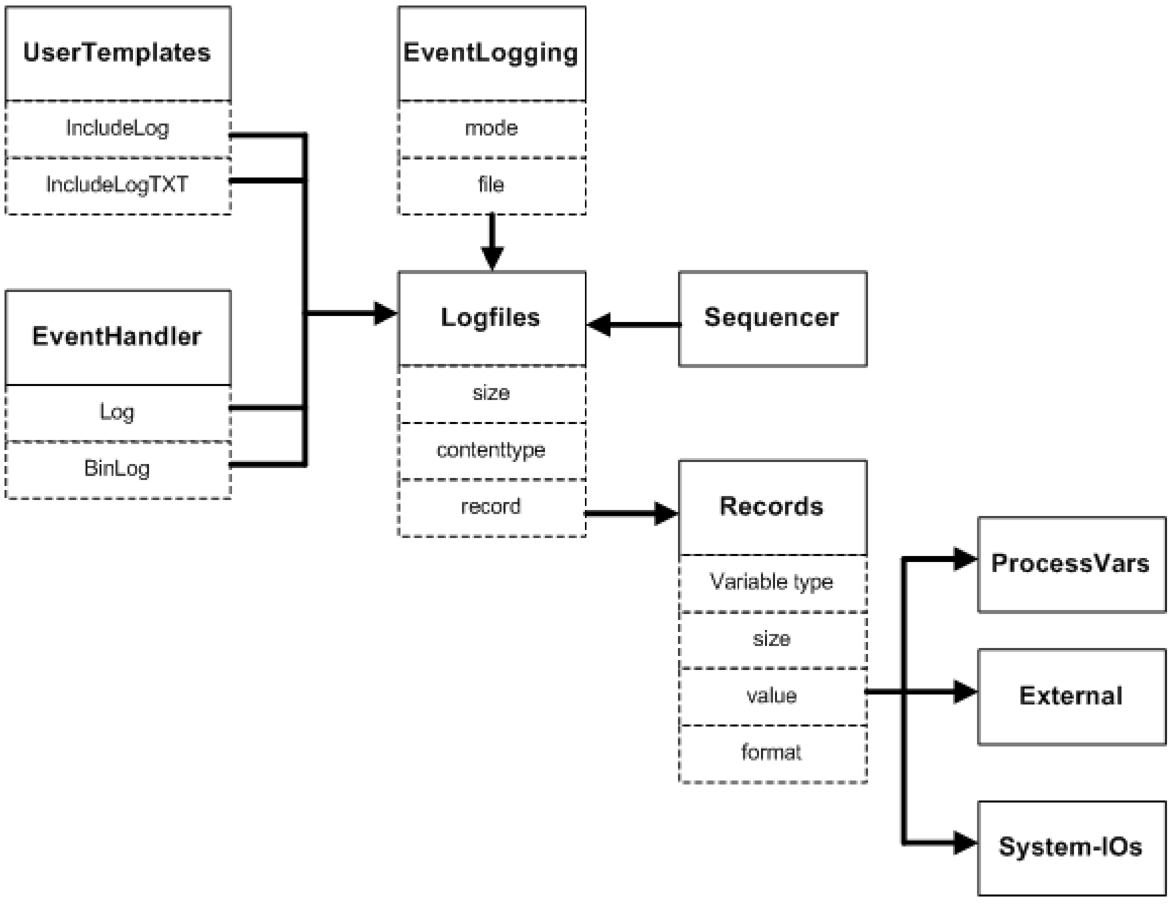
15.2 Event States, External, ProcessVars, System-IOs



15.3 MessageJobTemplates, UserTemplates, AddressBook



15.4 Logfiles, Records, EventLogging



16. Firmware

16.1 Remote Firmware Update (TFTP)

Remote Firmware Update is only supported on FP Gateways with SD-Card (Hx4xx).
A SD-Card with FAT16 file system must be mounted to upload the new firmware using TFTP.
The firmware file must be registered in the TFTP configuration (see Webserver TiXML manual for more information on TFTP configuration).

```
[<SetConfig _="ISP" ver="y">
  <TFTP>
    <Port _="69"/>
    <Files>
      <Upload _="0:Tixi.Gate_FW.tag.gz " acc="RW" size="10000000"/>
    </Files>
  </TFTP>
</SetConfig>]
```

Upload the "Tixi.Gate_FW.tar.gz " binary firmware file via TFTP using remote file name `0:Tixi.Gate_FW.tar.gz` ("0" addresses the SD-Card / USB memorystick).
You might use F: instead of 0: in case you don't use an SD-Card or USB memorystick.
After uploading, a `[<Reset _="Update"/>]` is necessary to flash the firmware.

16.2 Compatibility

The FP Gateway firmware will be updated periodically to implement new features, support more PLCs or for bug fixing. In most cases it's not necessary to update your projects after updating the firmware.

In the past, some changes have been made to the system structure to make it easier to understand. In most cases, a new firmware version supports both the old and the new structure.
Due to these changes, a project written for a new firmware may not work with an older firmware.

Index