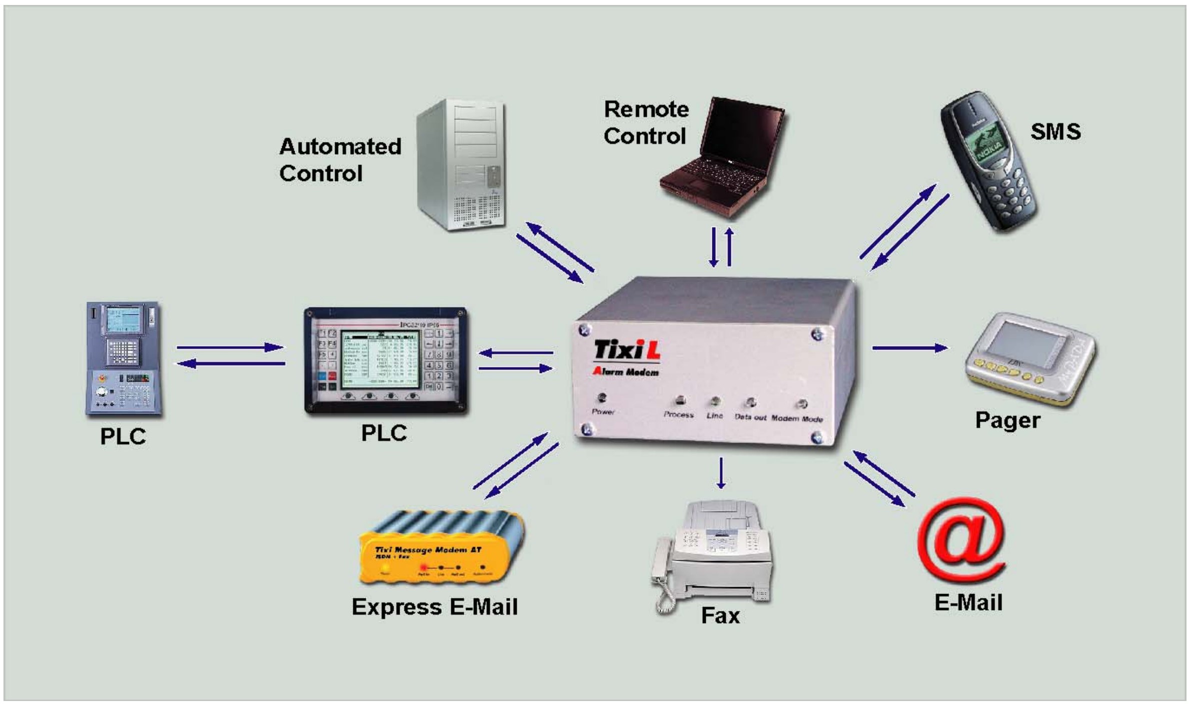
TiXML-EN



TiXML Reference Manual

for Tixi Alarm Modem
Aluline (all models) or
Hutline Hx20/Hx21/Hx23/Hx25/Hx27/Hx41/Hx47/Hx71/Hx76

V 2.2.74

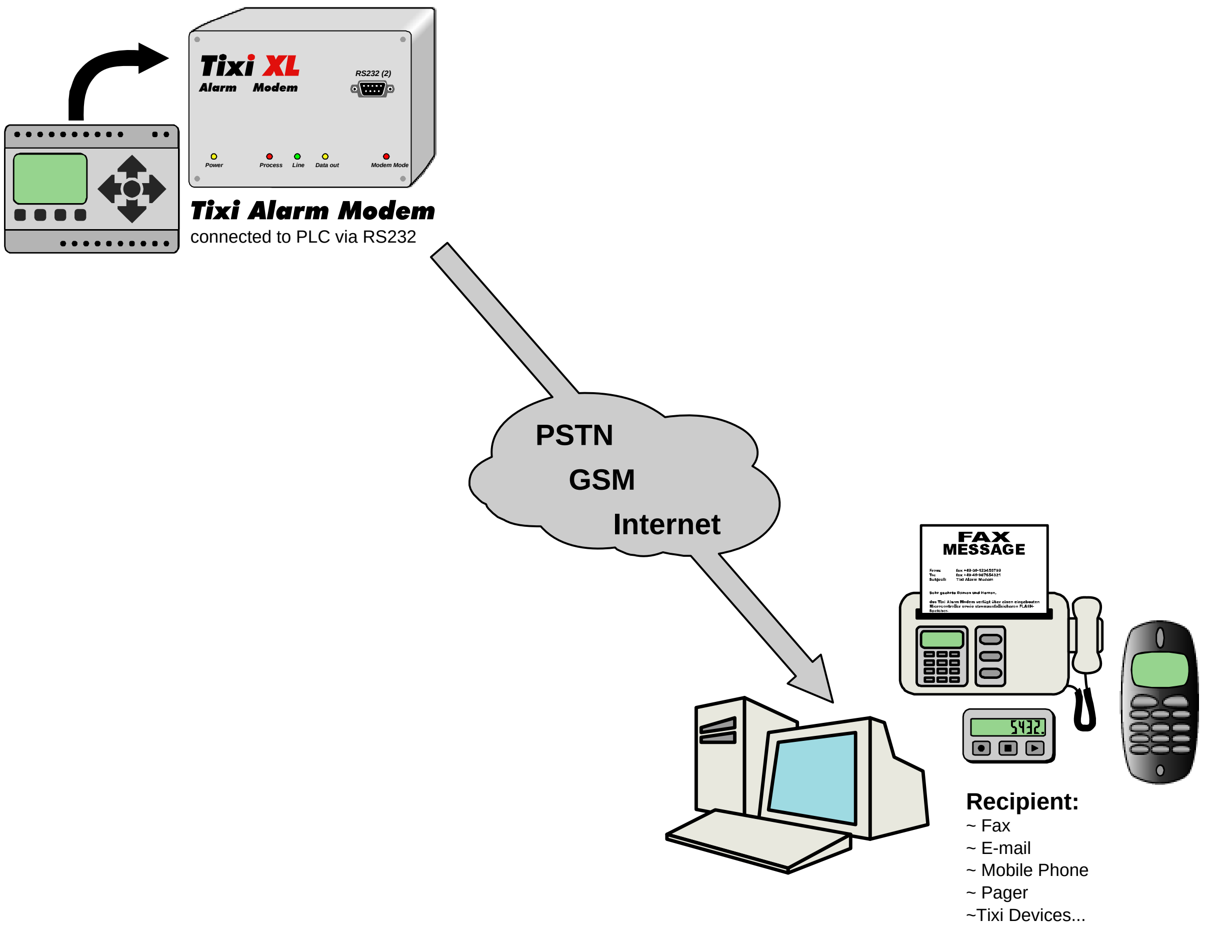
Publication close: 23. January 2008. Supported Tixi Alarm Modem Firmware: 2.02.74.0

1 Overview

The **Tixi Alarm Modem** provides you with a completely new type of communication device, which can be integrated into existing systems with ease.

The communication protocols of common PLCs are already implemented into the Tixi Alarm Modem, so there's no need to change the PLC or it's programming. Other PLCs can control the Tixi Alarm Modem via simple text strings: the **TiXML-commands**.

Imagine this as a simple application of Tixi Alarm Modem:



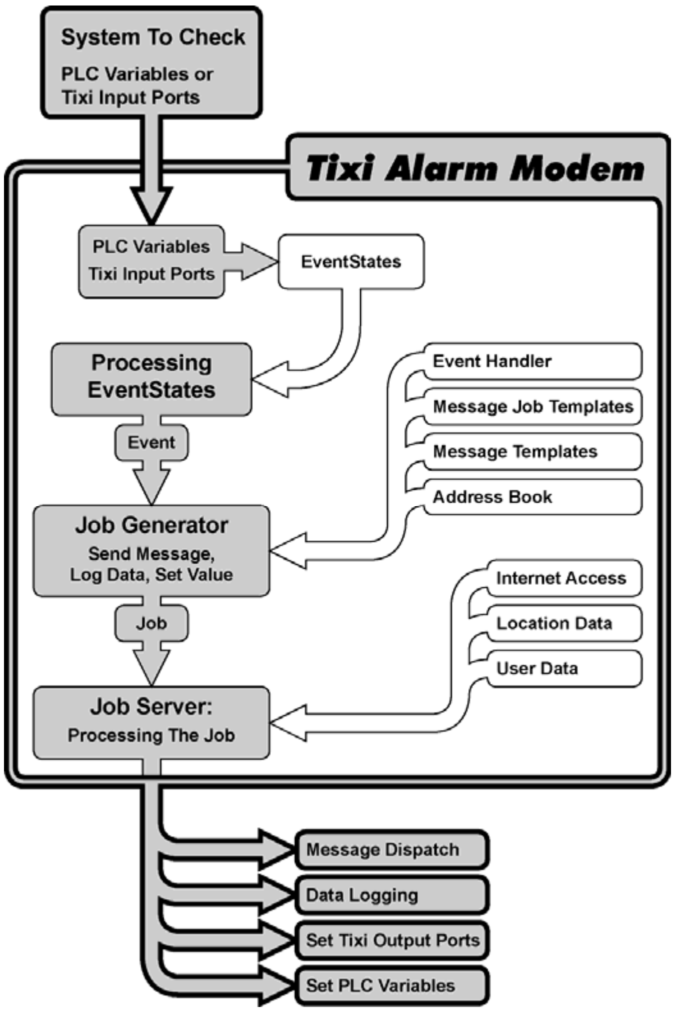
This manual describes the features and functions of the Tixi Alarm Modem that can be programmed with TiXML-Databases via TICO “TiXML-Console".

The development of the Tixi Alarm Modem firmware never stops, therefore some of the described features may not be available with your “older” hard- or firmware version. The cover tells you which firmware is supported by this manual version (Note “Supported Tixi Alarm Modem Firmware x.x.x”).

Besides some functions a small note “@FWx.x.x” tells you which major firmware release is necessary to use this feature. Without note means @FW 1.72.14.0 (first release firmware). See also chapter 14 for a detailed firmware history.

This manual is for the hardware models listed on the cover only. A TiXML-Reference for products with a firmware version 3.x (HMxxx/HGxxx/HExxx/HWxxx, all with three digits!) is available on our website (manual code “TiXML3-EN”).

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



At first there is a **“system to check"** which may be a **PLC** connected via serial port or some switches and measure instruments connected to the Tixi Alarm Modem **I/O-Ports**.
The configuration of PLCs is documentated 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 Tixi Alarm Modem via TiXML-commands, e.g. the DoOn-Command to activate alarms (see. **chapter 2.4.9.1**)

The process **“Event States"** are defining what to do if a variable or I/O-port changes (**chapter 6.3**). 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 0)** is active by an event state, the **“Job Generator"** starts to process the event handler commands e.g. for logging data (chapter 4) or creates an “Sendmail" job using the predefined text templates and addressbook contacts. A **“Message Job Template" (chapter 3.7)** defines the message type (e.g. SMS) and refers to the recipient from the **addressbook** (**chapter 3.4**) and the message templates (**chapter 3.6**).

Thereafter the “**Job Server**" starts to send the alarm message using the **location (chapter 0)** 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 internat mail servers. For SMS the the database of **SMS providers (chapter 0)** are used.

2 Controlling Tixi Alarm Modem

2.1 Overview

Tixi Alarm Modem has a serial interface (RS232). This interface is used to control **Tixi Alarm Modem** by a client (control unit, PC, Laptop etc.). There are two control protocols provided by **Tixi Alarm Modem**. Each is used in its related working mode. The Modem Mode uses the well-known AT command set to control Tixi Alarm Modem, while the TiXML - Control Protocol is used to control and configure Tixi Alarm Modem as a messaging system. TiXML is used to control Tixi Alarm Modem remotely via a phone line or via the Internet.

2.2 Modem Mode versus TiXML Mode

Tixi Alarm Modem has two working modes - the Modem Mode and the TiXML Mode. Only one of these modes can be used at a time.

Modem Mode
In this mode Tixi Alarm Modem works as an industrial AT modem. A client (for example the control unit) can use this mode to prepare custom messaging or remote control functions. This mode is indicated by the Modem Mode LED. Tixi Alarm Modem now responds to the AT command set. To quit the Modem Mode and enter the TiXML Mode send the command

```
AT+T Mode="TiXMLMode"
```

(see AT commands in chapter 2.3).

Note: In Modem Mode Tixi Alarm Modem is still processing the configured events! A created message will be queued and sent after Tixi Alarm Modem is switched back into TiXML Mode.

TiXML Mode
In this mode Tixi Alarm Modem works as a messaging system. A client (for example the control unit or a PC) can now send commands and configurations to Tixi Alarm Modem and can receive responses. Simply use a terminal program to do this. In this mode Tixi Alarm Modem is responsible for the Simple Tixi Control Protocol. To quit the TiXML Mode and enter the Modem Mode send the ‘Switch' command.

`[<Switch _="ModemMode"/>]` (see TiXML later)

or (only available directly after opening the com port, @FW 2.0):

```
AT+T Mode="ModemMode"
```

TiXML uses serial communication at any baudrate with data format 8N1. Hardware handshake (RTS/CTS) is recommended to transfer large databases.

2.3 AT Commands

When you are using Tixi Alarm Modem in the Modem Mode, it operates using the AT command set. In this mode it works like an industrial modem. The AT commands are described in the "Tixi Modem Manual" which is not a part of this document and is available from the download area of www.tixi.com.

To change the working mode of Tixi Alarm Modem an additional command is implemented, which sets the mode of Tixi Alarm Modem:

Command	***Values***	***Description***
AT+T Mode="*mode*"	*mode:* ModemMode TiXMLMode	Set the working mode of the Tixi Alarm Modem. Example: Set the Tixi device to the TiXML Mode: AT+T Mode="TiXMLMode"

2.4 TiXML - Control Protocol (TiXML)

The Simple Tixi Control Protocol (TiXML) is designed for use of Tixi Alarm Modem as a messaging system in industrial applications. As clients cannot implement difficult multi-layer protocols like TCP/IP, Tixi 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 Tixi Alarm Modem. The TiXML protocol is reduced so that only the really necessary data about this 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.

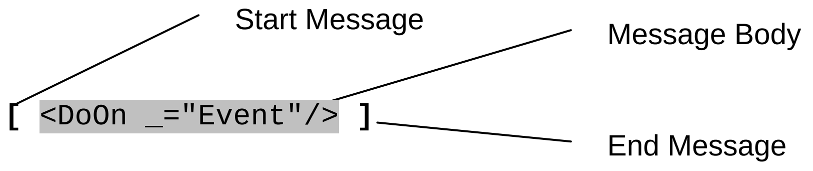
The protocol is derived from Simple Object Access Protocol (SOAP) [1] which is designed for message transports via HTTP and Internet to implement remote procedure calls via the Internet. In the TiXML the complete message envelope is replaced by a simple frame "[...]". The message contents (body) are used only. In future, the message envelope can be added to the body and the same protocol can be used to control Tixi Alarm Modem via the Internet. Like SOAP, the TiXML uses the Extensible Mark-up Language (XML) [2] 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 Tixi Alarm Modem (which is in the role of the server) and Tixi Alarm Modem answers with a return value (really a return message, containing more than one value).To call a procedure, a message is sent by the client to Tixi Alarm Modem. The message is enclosed by a message frame (see framing). The procedure and the parameter are encoded as an XML document (see Command Encoding).

2.4.2 Framing

Each Message is enclosed in brackets:



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.

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

Start Message

Message Body

End Message

The framing is the same for messages to Tixi Alarm Modem as for messages from Tixi Alarm Modem to the client.
Tixi Alarm Modem 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 accepted.

2.4.3 Command Encoding

Each procedure call and corresponding answer is encoded as an simple XML document. An 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 goes from the client to Tixi Alarm Modem the message is interpreted as a procedure call. In the opposite direction the message is the answer to the 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 Tixi Alarm Modem.

The client should wait with a timeout of 10s for an answer from Tixi Alarm Modem before it makes the next procedure call.

To generate something like an “AT command set" for controlling a Tixi Alarm Modem, 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

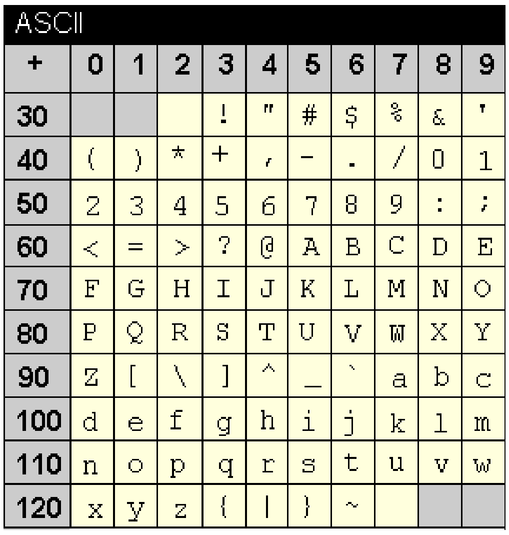
Children of the Root

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

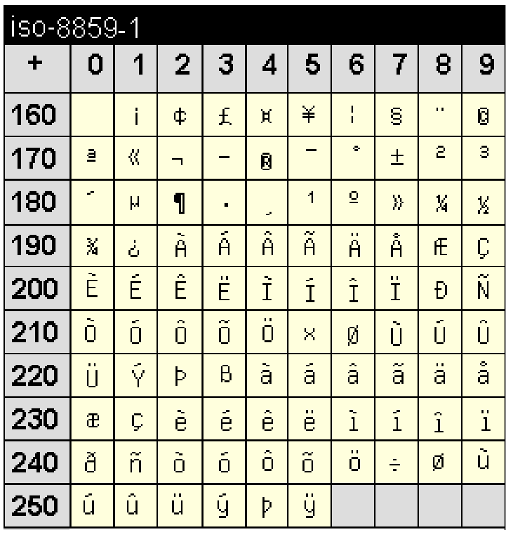
Only one <> element is allowed per line.

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":

Character	Entity
<	`&lt;`
>	`&gt;`
&	`&amp;`
"	`&quot;`

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

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

Tixi Alarm Modem responds: `[<DoOn/>]`

Please note that a command name or tag name must not be more than 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 the 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 a 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 an Parameter with the name 'Barn' and the value '12'.

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

Optional Parameter

Parameter List

If you have complex parameters you can encode it as an 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 modem, we recommend to stop the job processing before sending the first SetConfig:

```
[<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 modem reset):

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

2.4.4 Error Frame

When the command processing produces an error, Tixi Alarm Modem 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 Tixi Alarm Modem.

Elements:

e:
n...short error message - returns an error code **(default)**.
y…verbous error message – returns a short description
v...extended error message – returns an extended verbous 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 standard 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.

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 configdata 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 tagname 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
-300	modem connection failed
-306	error writing a read only variable
-307	build up PPP stack failed
-399	modem not connected
-401	framestream error
-510	file does not exist
-516	select function for send timed out
-520	ordered filespace can't be reserved
-607	stack level incomplete on exit
-1095	command is remotely not available
-1097	connection lost - still in remote mode
-1098	authentification 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 occured
-1899	event not in list
-2093	CHAP error - no default user defined
-2094	modem connection lost
-2095	no modem connection
-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 remotly not allowed
-2299	invalid reset command
-2391	comport used by configuration tool
-2397	the requested comport does not exist
-2491	no authentification method given
-2493	invalid authentification method
-2497	the user/password is invalid
-2698	error during reading the logfile
-2699	the log entry range is invalid

2.4.6 Commands

The TiXML implements some 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:

- Controlling the device
- Configuration
- Client Event Processing
- Testing

2.4.7 Controlling the Device

2.4.7.1 Switch

***Switch** - Set the working mode*

Syntax:

```
<Switch _="n"/>
```

Description:

This command sets the mode of the device. Currently the Modem Mode can be set from TiXML Mode.

Note: When the TiXML Mode is activated, **no answer** comes from the Tixi Alarm Modem and the command is processed.

Parameter:

n:
ModemMode....Modem Mode Tixi Alarm Modem responds to the AT command set.
TiXMLMode..........TiXML Mode: Tixi Alarm Modem responds to the TiXML.

Return:

If no error (command is processed):

Switch from TiXMLMode to the TiXMLMode itself:

```
<Switch/>
```

Switch from TiXMLMode to the Modem Mode:

no response.

On error (command is not processed):

see default error frame (chapter 2.4.4)

Example:

Set Tixi Alarm Modem into the Modem Mode.

Client sends:	**`[<Switch _="ModemMode"/>]`**
Tixi Alarm Modem responds:	**no answer, Modem Mode LED goes on.**

2.4.7.2 Reset

In addition to the TiXML reset command the device can be restored to its factory setting via the hardware:

1. unplug the power supply
2. keep the service button pressed, meanwhile plug in the power supply (the power LED should now flash)
3. release the service button and press it again until the power LED starts flashing very fast.
4. the device will now restart with factory settings (GSM seetings will be kept to guarantee remote access).

***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 Tixi Alarm Modem. Depending on device (especially Flash Memory Size) 6 s to 30 s will elapse before the device is ready again.

Parameter:

Mode:

Keep	Keep the current settings.
Factory	Sets the permanent memory of the device to its factory settings. **Note**: All configurations set with the 'SetConfig' command are deleted. GSM settings will be kept to guarantee remote access.
Process	Resets the Process (see Processing I/O Ports): All event states are set to idle and the current state of the input ports is read.
Download	Sets the Modem into firmware update mode. The command has to be followed by ATI9 <enter> to detect the upload baudrate (answer: “Serial mode, no modem code!”) Thereafter the binary firmware file may be sent using Z-Modem protocol.
number:	A magicnumber is necessary for the factory reset of a remote device only. The default 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 Tixi Alarm Modem and keep the current settings.

Client sends: **`[<Reset _="Keep"/>]`**
Tixi Alarm Modem responds: **`[<Reset/>]`**

2.4.7.3 Login

***Login** – Start a controlling session*

Syntax:

```
<Login _="type" user="User Name" password="Password"/>
```

Description:

This command starts a controlling session for a user. By this command the user makes an authentication for the device.

Note: The Tixi Alarm Modem can be protected against unauthorized access. In the factory configuration no access protection is provided. In this system state 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 but a controlling session is established which does not end with the command.

To limit the access to certain users you can create a user-password map in the configuration of the Tixi Alarm Modem. 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 or phone line connection for remote control) where 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 dialup connection is broken (for remote control only).

To create the user-password map use the `SetConfig` command (see "Set Login Data" chapter 3.11).
There are two ways to remove the access limitation:

1. Write an empty user-password map into the configuration command (see "Set Login Data" chapter 3.11).
2. Make a factory reset using the **`Reset`** command with the parameter `Factory`. See chapter 2.4.7.2 for details on resetting Tixi Alarm Modem. Bear 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 **(for feature extension).**

User Name User name can be empty if no authentication is required or it must be empty if the user "Default" is configured (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"/>]`**
Tixi Alarm Modem returns: **`[<Login/>]`**

Login not successful:
Client sends: **`[<Login _="PAP" user="Name" password="try"/>]`**
Tixi Alarm Modem returns: **`[<Error _="-1098"/>]`**

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

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

Tixi Alarm Modem returns:

```
[<Login _="Challenge">
<key _="b0a12c96ef9b01f8c07fd98b332c165ffdab5764872ef049" />
<id _="31" />
</Login>]
```

Client sends: `[<Login _="Response" id="31" md5="7b813bf46f6d3c2bdc62be64abdc001b" ver="y"/>]`

Tixi Alarm Modem returns: `[<Login/>]`

The hash for the MD5 response is calculated over the string "ID+password+key". The ID has to be converted into ASCII, e.g. ID=31 is decimal=49 is ASCII=1.
The string in the above example would be
`"1testb0a12c96ef9b01f8c07fd98b332c165ffdab5764872ef049"`

2.4.7.4 Logout

Logout *– Quit a controlling session*

Syntax:

```
<Logout/>
```

Description:

This command quits a controlling session - started with a successful Login. It deletes the access right to Tixi Alarm Modem for the connection where the command is sent.

Note: This command can be sent at any time. It affects the access to the Tixi Alarm Modem if a Login command is sent beforehand. In this case the access right is deleted. There is a new Login necessary to get access to the device. If no access protection is established (see Login command description) 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/>]`**

Tixi Alarm Modem returns: **`[<Logout/>]`**

2.4.7.5 SetTime

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 Tixi Alarm Modem to the value of the parameter.

Note: The Tixi Alarm Modem contains a Real Time Clock 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 Tixi Alarm Modem system time to 18 Aug 2000 13:10 hour and 34 seconds.

Client sends:	**`[<SetTime _="2000/08/18,13:10:34"/>]`**
Tixi Alarm Modem responds:	**`[<SetTime/>]`**

2.4.7.6 GetTime

***GetTime** - Gets the current system time of the device*

Syntax:
`<GetTime/>`

Description:
This command returns the current system time of the Tixi Alarm Modem.

Note: This command can be used to check whether the Tixi Alarm Modem 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 Tixi Alarm Modem.

Client sends:	**`[<GetTime/>]`**
Tixi Alarm Modem responds:	**`[<GetTime _="2000/08/18,13:10:34"/>]`**

2.4.7.7 GetJob

***GetJob** – Shows a list of currently active jobs*

Syntax:
```
<GetJob del="Mode"/>
```

Description:
This command shows a list of jop 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
	n	don't delete active jobs (default)
	mem	system internal, show memory state
	port	system internal, show TCP port state
	dcu	system internal, show DCU state
	route	system internal, show routing table

Return:

If no error (command is processed):

```
<GetJob>
   <JobGroup _="State">
      <Job_X>
         <Time _="UNIX"/>
         <Type _="Type"/>
         <Priority _="Priority"/>
      </Job_X>
   </JobGroup>
   …
</GetJob>
```

JobGroup: Name of job group.
Currently known groups:
```
Modem_Mode
Default
Express_E-mail_Send
Express_E-mail_Recv
```

```
                SMS_Receive
                SMS_Send
                POP3_Client
                HTTP_Server_In
                HTTP_Server_Out
                Time_Client
                URL_Send
                SMTP_Client
                Text_Fax
                Script_Send
                Incoming_Call
                Job_Result_Processor
                Remote_ModemMode
```

State: State of job group

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

X: Active job number (increases)

UNIX: Start time of job in UNIX format (seconds since 1.1.1970)

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

Priority: Priority of job (see chapter 3.8.1)

On error (command is not processed):

see default error frame (chapter 2.4.4)

<u>Examples:</u>

GetJob on idle system state:

Client sends: **`[<GetJob/>]`**

Tixi Alarm Modem responds:

```
<GetJob>
   <Modem_Mode _="Started"/>
   <Default _="Started"/>
   <Express_E-mail_Send _="Started"/>
   <Express_E-mail_Recv _="Started"/>
   <SMS_Receive _="Started"/>
   <SMS_Send _="Started"/>
   <POP3_Client _="Started"/>
   <HTTP_Server_In _="Started"/>
   <HTTP_Server_Out _="Started"/>
   <Time_Client _="Started"/>
   <URL_Send _="Started"/>
   <SMTP_Client _="Started"/>
   <Text_Fax _="Started"/>
   <Script_Send _="Started"/>
   <Incoming_Call _="Started"/>
   <Job_Result_Processor _="Started"/>
   <Remote_ModemMode _="Started"/>
</GetJob>
```

GetJob with SMS in message queue, waiting for acknowledge:

Client sends: **`[<GetJob/>]`**

Tixi Alarm Modem responds:

```
<GetJob>
   <Modem_Mode _="Started"/>
   …
   <Text_Fax _="Started"/>
   <Script_Send _="Started">
      <Job_3>
         <Time _="1107674129"/>
         <Type _="5"/>
         <Priority _="1"/>
      </Job_3>
   </Script_Send>
   <Incoming_Call_Trigger _="Started"/>
   <Job_Result_Processor _="Started">
      <Job_3>
         <Time _="1107674311"/>
         <Type _="65"/>
         <Priority _="99"/>
      </Job_3>
   </Job_Result_Processor>
   <Remote_ModemMode _="Started"/>
</GetJob>
```

GetJob delete:

Client sends: **`[<GetJob del=”y”/>]`**

The Tixi Alarm Modem answers with the list of all active jobs including the deleted jobs. A thereafter immedialtely send [<GetJob/>] will show a list without active jobs (if no new jobs are started in the meantime).

2.4.7.8 Remote Command

***Remote** - Connect to a remote Tixi Alarm Modem*

Syntax:

`<Remote aprot="`*`type`*`" number="`*`RemoteNumber`*`" user="`*`User Name`*`" password="`*`Password`*`"/>`

Description:

This command establishes a controlling connection to the remote Tixi Alarm Modem.

1. The phone number is used to establish a dialup connection to the remote device.
2. A Login is done for the connection.
3. The local Tixi Alarm Modem is switched to the remote connection mode.

Note: This connection can be closed by a Logout command, so the remote Tixi Alarm Modem hangs up.

The remote connection mode of the local device must be closed by the RemoteEnd command before the configuration of the local Tixi Alarm Modem can be continued.

Parameter:

type:

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

number:

One of the following two parameters must be present.

phone

the following parameter is the canonical phone number in the international format (+49-30-40608500). In this case, the dial properties configured in the 'Location' group of the USER database (see chapter 0) are used to create the modem dial string.

or

dial

the following parameter is the modem dial string which should be used to establish the dialup connection. In this case the settings in the Location group are ignored.

RemoteNumber

The meaning of this value depends on the parameter name:

parameter name is **phone**:

The phone number in international format of the remote Tixi Alarm Modem (e.g. `+49-30-40608500`)

parameter is **dial**:

The value is the command string for the modem which is used to dial to connect to the remote Tixi Alarm Modem at the current phone port where the local Tixi Alarm Modem is connected to (for example: `ATDT0,003040608500`)

User Name:

User name for the login of the remote Tixi Alarm Modem. It could be empty if there is no access protection or a password protection is configured in the remote device **(default empty)**.

Password:

Password for the login of the remote Tixi Alarm Modem. It could be empty if there is no access protection configured in the remote device or the user's password is empty **(default empty)**.

Return:

If no error (command is processed):

```
<Remote/>
```

On error (command is not processed):

see default error frame (chapter 2.4.4)

Example:
Connect to a remote Tixi Alarm Modem without login using the configured dial parameters.
Client sends: **`[<Remote phone="+49-30-4060821"/>]`**
Tixi Alarm Modem responds: **`[<Remote/>]`**

Connect to a remote Tixi Alarm Modem without login using the dial string.
Client sends: **`[<Remote dial="ATDT021"/>]`**
Tixi Alarm Modem responds: **`[<Remote/>]`**

Connect to a remote Tixi Alarm Modem without login using the dial string and preparing the connection for a later connection to the remote client device with a client device control program:
Client sends: **`[<Remote dial="AT&#x26;D0DT021"/>]`**
Tixi Alarm Modem responds: **`[<Remote/>]`**

Connect to a remote Tixi Alarm Modem with login using the dial string and the authentication protocol CHAP:
Client sends: **`[<Remote dial="ATDT021" aprot="CHAP" user="admin" password="secret"/>]`**
Tixi Alarm Modem responds: **`[<Remote/>]`**

Controlling a Tixi Alarm Modem remotely:

1. Connect and login to the remote device:
Client sends: **`[<Remote dial="ATDT021" aprot="CHAP" user="admin" password="secret"/>]`**
Tixi Alarm Modem responds: **`[<Remote/>]`**

2. Send TiXML commands to control the remote device.

3. Disconnect:
Client sends: **`[<Logout/>]`**
Tixi Alarm Modem responds: **`[<Logout/>]`**

4. End the remote session.
Client sends: **`[<RemoteEnd/>]`**
Tixi Alarm Modem responds: **`[<RemoteEnd/>]`**

***RemoteEnd** - Exit the remote connection mode of the local Tixi Alarm Modem*

Syntax:
`<RemoteEnd/>`

Description:
This command is to be sent after a successful Remote command when the connection is finished. This signals the local Tixi Alarm Modem that the following commands sent after this command should be processed by the local Tixi Alarm Modem. After disconnect all commands will be rejected by the local Tixi Alarm Modem as long as no RemoteEnd command is sent. This prevents the local device from receiving commands intended for the remote device when the connection is lost unexpectedly.

Parameter:
No Parameter.

Return:

If no error (command is processed):

```
    <RemoteEnd/>
```

On error (command is not processed):

see default error frame (chapter 2.4.4)

Example:

Connect to a remote Tixi Alarm Modem without login using the configured dial parameters:

Client sends:	**`[<Remote phone="+49-30-4060821"/>]`**
Tixi Alarm Modem responds:	**`[<Remote/>]`**

Do some remote configurations.

Logout to disconnect the connection:

Client sends:	**`[<Logout/>]`**
Tixi Alarm Modem responds:	**`[<Logout/>]`**

Exit the remote connection Mode of the local Tixi Alarm Modem.

Client sends:	**`[<RemoteEnd/>]`**
Tixi Alarm Modem responds:	**`[<RemoteEnd/>]`**

2.4.7.9 TransMode Command

***TransMode** - Set the remote Tixi Alarm Modem to the Modem 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 Tixi Alarm Modem 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 extension com port or the host port (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 Tixi Alarm Modem is finished when the dialup connection drops down. After this the remote Tixi Alarm Modem goes back in the TiXMLMode. To drop the connection the local desktop PC must send the escape sequence (+++) and ATH, if the TransMode command was issued over the modem.
A transparent mode to the host port MB (COM1) is blocked if a local login session is open (see chapter 2.4.7.3).

If the TransMode was issued through a local modem (e.g. from COM1 to COM2), the Modem goes back to TiXML Mode after DTR-low (keep parameter) on COM1 or after a PnP initialization.

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 9600).**

comport:

Specifies the COM port on the Tixi Alarm Modem used for the connection.

Hutline Modems:

`COM1`	Programming port (labeled ***COM1 RS232***) (**default**)
`COM2`	PLC port (labeled ***COM2 RS232*** or ***COM2 R-485/422***) (if available)

Aluline Modems:

`COM1`	mainboard port (labeled ***RS232(1)***) (**default**)
`COM2`	port on extension board #0 (labeled ***RS422/485)***) (if available)
`COM3`	port on extension board #1 (labeled ***RS232(2)***)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

Note: RS 485/422 communication is only possible with special RS 485/422 interfaces.

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

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

During connection from one Tixi Alarm Modem to another (optional):
Specifies the time period Tixi Alarm Modem will try to disable the bus protocol at the remote Tixi Alarm Modem to establish a transparent connection.

timeout:
Specifies the time the Tixi Alarm Modem 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 Tixi Alarm Modem to the transparent mode and connect it to the client device with 57600 bbs 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"/>]`**
Tixi Alarm Modem responds: **`CONNECT`**

2.
Connect to a remote Tixi Alarm Modem, switch the device 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"/>]`**
Tixi Alarm Modem responds: **`CONNECT`**

1. **Disconnect the client (R-CON, TILA2, TICO) from Tixi Alarm Modem COM Port.**
2. **Connect the other control program (e.g. PLC software) to the COM Port.**
3. **Control the remote client.**
4. **Disconnect the control program from COM port.**
5. **Connect the client (R-CON, TILA2, TICO) to the Tixi Alarm Modem COM port.**
6. **Disconnect:**
   Client sends: *(wait one second)* **+++** *(wait one second)*
   Tixi Alarm Modem responds: **`OK`**
   Client sends: **`ATH`**

2.4.8 Configure the Device

2.4.8.1 SetConfig

***SetConfig** - Set configuration data.*

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

Description:
This command writes configuration data into a database.

Note: Tixi Alarm Modem stores this data permanently in an embedded XML - database. The database is prepared by the firmware of Tixi Alarm Modem. 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/changed (**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 1:

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>]
```

Tixi Alarm Modem responds: **`[<SetConfig/>]`**

Example 2:
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>]
```

Tixi Alarm Modem responds: **`[<SetConfig/>]`**

2.4.8.2 GetConfig

GetConfig *- Get configuration data.*

Syntax:
```
<GetConfig _="Path"/>
```

Description:
This command reads the configuration data from a database of the Tixi Alarm Modem

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/changed (**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"/>]`**
Tixi Alarm Modem responds:
```
[<GetConfig>
   <ISP>
      <PPPComm>
         <PPPUserName _="user"/>
         <PPPPassword _="pass"/>
         <AuthentFlags _="3"/>
         <FirstDNSAddr _="194.25.2.129"/>
         <SecondDNSAddr _="193.158.131.19"/>
      </PPPComm>
```

```
                    <SMTP>
                       <mailserver_name _="tixi.com"/>
                    </SMTP>
                    <Modem>
                       < RemotePhoneNumber _="+49-30-40608117"/>
                       <MediaType _="DATA"/>
                       <ModemProtocol _="syncPPP"/>
                    </Modem>
                 </ISP>
              </GetConfig>]
```

2.4.8.3 Get Value

***Get** - Get System Property*

Syntax:

```
<Get _="Path" AddInfo="Error" format="FormatString"/>
```

Description:

Get the value of the system properties referred by the *Path* value.

The System Properties are the set of data describing a Tixi Alarm Modem. 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 also 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 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 UserTemplates or the EventHandlers which could contain some elements with the same name. 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, Tixi Alarm Modem will send a list of all system properties.

Path:

Path which addresses the system property. See Appendix - System Properties for details on system properties.

Error : (@FW 2.2)

For directly reading the error state of a 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.
See PLC-TiXML-Manual for further information.

FormatString: (@FW 2.2)

integer: (for PLC and process variables)

1. **simpleType = Uint8, Uint16, Uint32, Int8, Int16, Int32** (see 6.7.1).
   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 6.7.1).
   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 6.7.1).

native: (or empty)
The value is displayed native. (see 6.7.1).

FormatString:
String that defines the value output format.
For a list of available format option see chapter 6.7.

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 (chapter 2.4.4)

Example:

Get the Tixi Alarm Modem serial number.

Client sends:	**`[<Get _="/SerialNo" ver="y"/>]`**
Tixi Alarm Modem responds:	**`[<Get _="00081101"/>]`**

Get the size of RAM in bytes.

Client sends:	**`[<Get _="/Hardware/RAM/Size"/>]`**
Tixi Alarm Modem responds:	**`[<Get _="524288"/>]`**

Get the state of the first digital input of the HutLine mainboard.

Client sends:	**`[<Get _="/Process/MB/IO/I/P0"/>]`**
Tixi Alarm Modem responds:	**`[<Get _="1"/>]`**

Get and reformat the state of the analog input of the HutLine mainboard.

Client sends:	**`[<Get _="/Process/MB/A/AI/P0" format="F,2"/>]`**
Tixi Alarm Modem responds:	**`[<Get _="31,45"/>]`**

Get the native value of a formatted process variable:.

Client sends:	**`[<Get _="/Process/PV/Variable" format="native"/>]`**
Tixi Alarm Modem 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"/>]
```
Tixi Alarm Modem responds: `[<Get _="0,0"/>`

Get the maximal dial attempts defined in the USER database USER section.
Client sends: `[<Get _="/USER/USER/MaxDialAttempts"/>]`
Tixi Alarm Modem responds: `[<Get _="2"/>]`

2.4.8.4 Set Value

***Set** - Set System Properties*

Syntax:
```
<Set _="Path" value="Value"/>
```

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 Tixi Alarm Modem. 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 UserTemplates or the EventHandlers which could contain some elements with the same name. 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.

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"/>]`**

Tixi Alarm Modem responds: **`[<Set/>]`**

Set the the first digital output of the HutLine mainboard.

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

Tixi Alarm Modem responds: **`[<Set/>]`**

Set a process variable defined with exp=”-2”.

Client sends: **`[<Set _="/Process/PV/Variable" value="3.12"/>]`**

Tixi Alarm Modem responds: **`[<Set/>]`**

2.4.9 Testing and Working

2.4.9.1 DoOn

***DoOn** - Process a client event.*

Syntax:

```
<DoOn _="EventName">
    ParameterList
</DoOn>
```

or

```
<DoOn _="EventName"/>
```

Description:

This starts the processing of a client event.

Note: There are two syntax forms, 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.

Typically, this leads to the creation of message job(s). The sending process of the messages is done while the client can create new event messages. The results of the processing are observed by Tixi Alarm Modem itself. The results can be observed by the client using the `ReadLog` command which is described in chapters 2.4.9.2 (on how to read logfiles) and 4 on how to create some.

Parameter:

EventName:

Name of the event to be processed. There must be an event handler configuration in the 'EVENTS' database which has this name. See chapter 3.2 for details on EventHandler database.

ParameterList:

List of XML encoded parameters representing the event context.. A parameter is written in a single XML tag with:

```
<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>]
```

Tixi Alarm Modem responds:

```
[<DoOn/>]
```

2.4.9.2 Reading and clearing logfiles

ReadLog *– Read entries from the system's log files.*

Syntax:

```
<ReadLog _="LogFileName" range="entryrange" type="templates"
flags="header" Formats/>
```

Description:

Tixi Alarm Modem 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 also allowed. 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 actual 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: (@FW 2.0)

Predefined logfile formats:

XML:	Logfile will be displayed as XML file
CSV*:*	“comma separated format”, 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" (only for templates CSV/HTML)

NoId:	removes the ID of each entry
NoDate:	removes the Date of each entry
NoTime:	removes the Time of each entry
NoNames:	removes the first row with variable names (@FW 2.2)

Formats: See chapter 4.9 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"/>

      …

   </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 of the logfile entry.

Time:

The creation time of the logfile entry.

Element:

Description of logged element, e.g. variable name of record definition.

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)

Range Examples:

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.

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

Get the entries within a timespan on a specific day.

```
[<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 actual 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 actual day.

Get entries from last 24h (smooth).

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

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

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"/>]`**
Tixi Alarm Modem responds: **`[<Clear/>]`**

3 Creating XML Projects

If you send complete projects or large databases to the modem, we recommend to stop the job processing before sending the first SetConfig:

```
[<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 modem reset):

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

3.1 Define Events

The typical application of Tixi Alarm Modem is the sending of messages initiated by an event which is signalled by the client sending a `DoOn` message to Tixi Alarm Modem. This message contains the event name and additional parameters describing the context of the event. The first step to use Tixi Alarm Modem this way is to define the events and to define which additional data should be sent with a message. We recommend the following procedure:

1. Define what the message should say
Usual messages that report the occurrence of an event have a similar structure:

```
    To: <RP@control.room.com>
    From: <ChickenFarm@Far.com>
    Subject: Barn is out of temperature
    Date: Thu, 16 Nov 2000 11:38:34 MET

    Barn is out of temperature ——————————————   Event Description

    Barn: 12                                    Measured Property
    Temperature of the Barn= 10 C ——————————

    Check the barn. ————————————————————————    'To Do' Message
```

Event Description:
Every sendmail event should be provided with a short message text describing the event. There should be a map from event names to the event descriptions.

Measured Property:
Typically, when an event occurs you have 'measured' some values describing the context of the event. Some of them can be determined by a sensor and some of them can be implicitly given by the address of the sensor in the field bus (which defines the location of the measured sample).
The example shows a list of two measured properties. Each of them has a name, a value and an optional unit. Name and unit can be defined by the message template stored in Tixi Alarm Modem where the value comes from the event message parameter list.

'To Do'Message:
Alert messages should particularly contain a clear instruction for the receiver of what he should do when he receives the message. In most cases it is related to the event.

Assuming the event message has this format, let's see which items the event message has to contain:

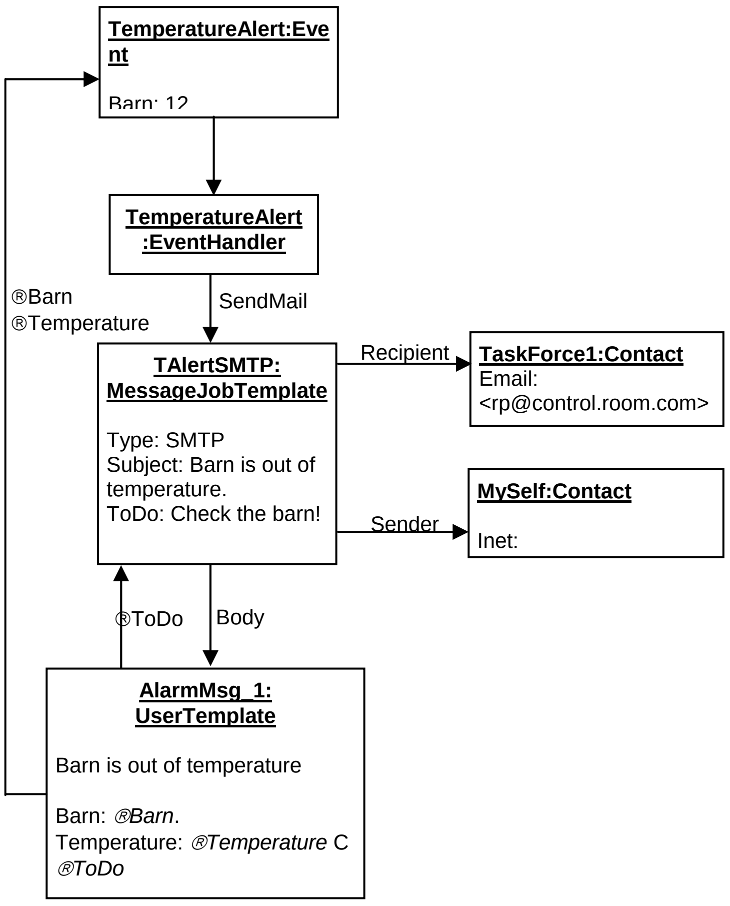
```
To: <RP@control.room.com>
From: <ChickenFarm@Far.com>
Subject: Barn is out of temp
Date: Thu, 16 Nov 2000
11:38:34 MET

Barn is out of temperature        [<DoOn _"TemperatureAlert">

Barn: 12                          <Barn _="12"/>
Temperature of Barn = 10 C        <Temperature _="10"/>
                                  </DoOn>]
Check the barn.
```

The residual items of the message are related to the event and defined by a message template.

When you know which message you would send to the recipients when an event occurs, you are ready to define the corresponding events.

2. Separate the Events
A frequent problem is to define what an event is and how it can be described by an event message. We recommend you first define which different event descriptions your system produces, and then relate an event for each of them. In most cases this leads to events with different contexts and this needs different message templates later in the event processing configuration. When you find some events with similar description and with the same context (but the values of the context can change from event to event) combine the events in one only. This simplifies template creation later in the configuration process.

The following example shows the dependencies between the configuration items the Job Generator uses to generate a job.



The event message (Event) has the name 'TemperatureAlert' and the parameters 'Barn' and 'Temperature'. The processing of the event is defined by the EventHandler 'TemperatureAlert' which has the name of the corresponding event.

The event handler contains the command SendMail that starts the Job Generator. The 'SendMail' command has a MessageJobTemplate as a parameter. This template - in this case with the name 'TAlertSMTP'- defines the job that the Job Generator should create.

The Message Job Template defines the type of the message job ('SMTP' is the type of an Internet e-mail), the recipient, the sender, the subject, the ToDo message which should be inserted in the message text and the body of the message (this is the text of the Internet e-mail).

The recipient and the sender are both defined as contacts in the AddressBook database (see chapter 3.4). In the example the Sender is named by 'MySelf' and the Receiver is named by 'TaskForce1'. The Body is a UserTemplate - in this case 'AlarmMsg_1'.

It defines the text of the message. The text contains some references (marked by ®) like Barn, Temperature and ToDo. When the message text is generated by the Job Generator, these references are replaced by the value of the corresponding parameter:

Barn: ®Barn $\longrightarrow$ Barn: 12
Temperature: ®Temperature C $\longrightarrow$ Temperature: 10 C

®ToDo $\longrightarrow$ Check the barn!

When the Job Generator has created the job, it adds the job to the Job Servers job list and sends an answer message on the DoOn command to the client. If there are some errors in the configuration of the job generation, the corresponding error message is returned and the job is not created.

The Job Server schedules the job and starts it when the internal modem is available to send the message. It uses the Location data - describing the dialling properties - to build the dial string and it uses the ISP data to connect to the Internet. If the connection fails, the job is automatically repeated by the Job Server in accordance with the redial attempts and SendMail command settings. (see chapter 3.2 on SendMail command parameters.)

The following chapters describe the steps to create and test an event handling configuration in detail.

3.1.1 Handling references

The great advantage of the XML databases is the possibility of linking 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 entity &#xae;) followed by the path to the value.

Depending on the location in the database, you have to use a different reference path.

Some 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/entry;"`

  or MessageText to another MessageText:

  `"&#xae;/D/UserTemplates/Signature;"`

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

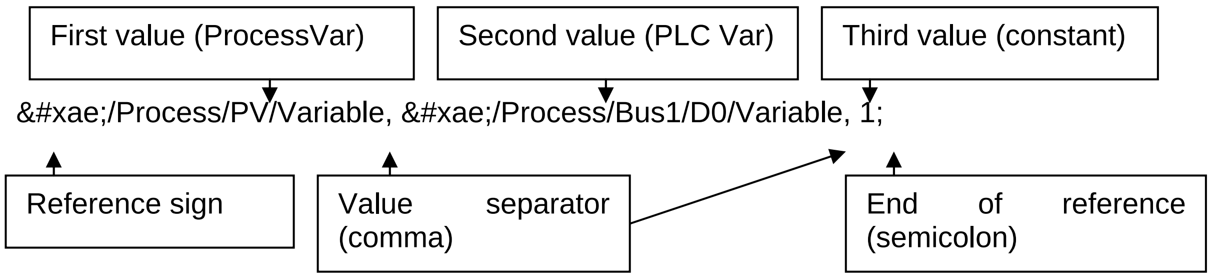
  Example:
  Reference from the TEMPLATE database to USER database:

  `"&#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/TemplateName"` (without reference symbol !)

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



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

3.1.2 Handling time settings

In the project structure 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 Configure Tixi Alarm Modem's User Data

Tixi Alarm Modem has a database called “USER" which contains some Tixi Alarm Modem configurations defined by the user of Tixi Alarm Modem. This includes the settings for Express E-Mail and the settings for call acceptance.

Database path: /USER/USER

```
<USER>
    <Handshake _="RTSCTS"/>
    <InitString0 _="ATX3M1L1"/>
    <ModemProtocol _="default"/>
    <IsdnDataChannelID _="*"/>
    <IsdnFaxChannelID _="*"/>
    <DChannelProtocol _="DSS1"/>
    <ModemListenMode _="OFF"/>
    <ModemParams _=""/>
    <BoxName _="Tixi Alarm Modem-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"/>
    <GSMModem _="C0"/>
    <GsmPorts _="0">
</USER>
```

Insert your own data

Name	***Description***
Handshake (@FW 2.2)	COM-Port Handshake for TiXML communication. **RTSCTS** Hardware Handshake (default) **XONXOFF** Software Handshake
BoxName	Name of Tixi Alarm Modem when it is used as Mail Box for Express E-Mail and as well in Fax message headlines.
BoxNumber	Canonical phone number of Tixi Alarm Modem. It identifies the phone network connection of Tixi Alarm Modem when it is used as a Mail Box. Syntax: ***CountryIDAreaCodeLocalPhoneNumber***
TimeZone	Time zone where Tixi Alarm Modem 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 3.8.1 on that.
RedialDelay	Time to wait between dial attempts in seconds **30s...255s** Timer starts after failed sendmail.
IsdnDataChannelID	Multiple Subscriber Number (MSN) for data calls. ***** answers on all numbers (default). **nn** MSN of the device (up to 16 digits)
RingCounter	Number of rings until Tixi Alarm Modem 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" required.
AccountExpiry (@FW 2.2)	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 (@FW 2.2)	Response parser for AccountQuery and AccountExpiry. Format: "amount:[word before cedit];valid:[word before expiry];format:[expiry format]"
Pin1	The PIN for the GSM – modem phonecard.
Pin2	A second PIN for the GSM – modem phonecard, if required.
GSMModem	Card address of the connected GSM modem. **0...15** (not necessary for Hutline GSM) **Note for Aluline:** If you use a Tixi Alarm Modem GSM without this setting, the Tixi Alarm Modem will use its incorporate V.90 or ISDN modem. A change of this value requires a restart of the Alarm Modem.
GsmPorts	Defines if the I/O ports of the incorporate GSM modem (Aluline only) are used as inputs or outputs. 0: both Inputs (default) 1: P0 Input , P1 Output 2: P0 Output, P1 Input 3: both Outputs Additionally to the software settings the hardware settings must be changed too.

Note: The GSM settings are not deleted by a factory reset ! To delete the GSM settings its necessary to overwrite the settings with empty values.

3.3 Configure the 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 Tixi Alarm Modem is installed.

During development define this location for the place where you will test the Tixi Alarm Modem. Later at the place where the Tixi Alarm Modem 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 Tixi Alarm Modem calls the proper number.

From our experience, defining a bad location is a most common error in configuring Tixi Alarm Modem. Please follow the following hints provided.

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

If no extension 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 Phone Number.
5. Insert the Dial Rules: typically `Tone,NoWaitForDialTone`.

Database path: /USER/Location

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

If an extension is used the configuration depends on the properties of the extension.

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 extension *can* define some prefixes you have to call in front of a phone number. Furthermore, some extensions define different prefixes for different call types, others define the same or nothing for the call types. Our location defines three call types:
   a). Internal Call (InternalDialPrefix): the call resides inside the extension.
   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. Extensions typically define a range of internal numbers, which can be called to get a person inside the extension, e.g. from the number 123456-200 you can dial 300 instead 123456-300. The last three digits are called Extension Number of the location where the phone number is the first five digits (the same for all locations inside the extension).

   If your extension defines internal (short) numbers, fill the 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 extensions only. But check whether your extension supports dial tone recognition or not.

How the Modem dials

There are several events where the Alarm Modem 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 modem 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.

<u>Example:</u>

1. The Modem 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 Modem 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 modem 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 modem 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 modem will dial “1 00 44 170 3333333” (long dial prefix, country prefix, country code, area code and phone number) because all settings are different.

2. The Modem 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 modem will only dial “! 111” (internal dial prefix !=FLASH, and extension number) because the CountryCode, AreaCode and phone number are the same.

Name	***Description***
CountryPrefix	Country prefix for international calls, e.g. 00 inside Germany.
CountryCode	Country code of location without prefix for international long distance calls, 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 prefix for domestic long distance calls, e.g. 30 for Berlin **GSM-Note:** If you are using a Tixi Alarm Modem 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	Extension prefix to receive a dial tone for local calls (to receiver with equal area code). May be blank.
LongDialPrefix	Extension prefix to receive a dial tone for long distance calls (to receiver with different area code). May be blank.
PhoneNumber	Local phone number of the location without any prefix or area code. If **no extension** is used this is the complete phone number (e.g. 123456500). If an **extension is used,** this is the phone number of the extension and the complete phone number is given by this field and the Extension Number. Example: From the outside of a company everyone has to call an eight digits number to get a certain person: 123456-500 (for the boss). 123456-501 (for Mr. Mayer) 123456-506 (for Ms. Greenspan)

	All numbers have the same six first digits: '123456' which is the value you have to insert in the field 'PhoneNumber'. In the field 'ExtensionNumber' insert the last three digits e.g. 500 (for the boss) 501 (for Mr. Mayer) 506 (for Ms. Greenspan)
InternalDialPrefix	Extension prefix to receive a dial tone for internal calls. May be blank.
ExtensionNumber	Last n digits of the phone number defined by the extension.
DialRules	Defines the rules to dial for the modem of the Tixi Alarm Modem. *DialMode*:*DialToneRecognition* ***DialMode (used for analog devices only):*** **Tone....**Tone dialling **Pulse....**Pulse dialling ***DialToneRecognition:*** **WaitForDialTone....**Wait for dial tone. **NoWaitForDialTone....**Do not wait for dial tone. e.g.: `Tone,NoWaitForDialTone` Use tone dialling and do not wait for dial tone.
NumberFormat (@FW 2.2)	Specifies how the modem will dial the recipients number ***** **Network related (default)** If it's a GSM modem, all numbers are dialed canonical. If it's a PSTN/ISDN modem, all numbers are dialed depending on the location settings. **n** **location related (PSTN only)** All numbers are dialed depending on the location settings (see "How the modem dials"). **c** **canonical (GSM only)** All numbers are dialed canonical (e.g. +49172123456).

3.4 Configure the Address Book

Each message created by Tixi Alarm Modem must include a sender and one or more receiver addresses. Tixi Alarm Modem provides an address book for all receivers and senders of messages. So if you create different messages which are sent to the same receiver 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 to use not more than 100 addressbook entries.

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

Database path: /TEMPLATE/AddressBook

```
<AddressBook>
   <MySelf>
      <Email _="user@domain.com"/>
      <Express-Email _="USER+49-30-1234567"/>
      <SMS_No _="+49-171-1234567"/>
      <Fax _="+49-30-1234567"/>
   </MySelf>
   <Addr1>
      <Email _="user2@domain2.com"/>
      <Express-Email _="USER2+49-30-1234567"/>
      <SMS_No _="+49-174-1234567"/>
      <SMS_Provider _="D2"/>
      <Fax _="+49-30-1234567"/>
      <CityRuf _="3949000"/>
      <Pager_Provider _="CityRuf"/>
      <URL _="https://www.devicecontrolnet.com/notification/">
      <URLPort _="80"/>
   </Addr1>
</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 receiver 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 transports.

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

```
<MySelf>
   <Email _="1044-79@online.de"/>
   <Express-Email _="USER+49-30-1234567"/>
   <SMS_No _="+49-160-1234567"/>
   <SMS_Provider _="D1"/>
   <Fax _="+49-30-40608336"/>
</MySelf>
```

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

```
<Addr1>
   <Email _="user2@domain2.com"/>
   <Express-Email _="USER2+49-30-1234567"/>         Address entry
   <SMS_No _="+49-174-1234567"/>
   <SMS_Provider _="D2"/>
   <Fax _="+49-30-1234567"/>
   <URL _="https://www.devicecontrolnet.com/notification/">
   <URLPort _="80"/>
</Addr1>
```

Name	*Description*
Email	Internet address of the contact (e.g. user2@domain2.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). Note: Up to firmware 1.72.14.0 it has to be entered without country code and “+“ or “-“ signs, e.g. 01611234567. Starting with firmware 2.0 it may be entered in canonical format if the “NumberFormat” in the SMS-Provider is configured (see chapter 3.9).
SMS_Provider	Name of the SMS provider used when the address is a receiver address. In this case the related SMS dial in of the provider is used. The following SMS providers are prepared in our software and TiXML examples: **D2** SMS with D2 (analog Tixi Alarm Modem) **D2_ISDN** SMS with D2 (ISDN Tixi Alarm Modem) **D1** SMS with D1 (analog Tixi Alarm Modem) **D1_ISDN** SMS with D1 (ISDN Tixi Alarm Modem) **Eplus** SMS with E plus **Mobilkom_A_TAP** SMS with Mobilkom Austria **GSM** SMS with GSM **Telekom** SMS via PSTN (only for the JF- Tixi Alarm Modem) **AnnyWay** SMS via PSTN (only for the JF-Tixi Alarm Modem)
Fax	Fax telephone number of the contact (for example +49-30-1234567) written in the international phone number format: *+CountryCode-AreaCode-LocalPhoneNumber*
Express-Email	Express E-Mail address of the contact (for example USER+49-30-1234567). This is inserted into the header of Express E-Mail. It consists of the Tixi user name and the international phone number of the receiving Tixi-Mail Box or Tixi Super Modem. The international phone number format is: *+CountryCode-AreaCode-LocalPhoneNumber*
CityRuf	CityRuf number of the contact, for example 3949000.
Pager_Provider	Currently only ‘CityRuf’ is supported.
URL (@FW 2.0)	URL of the HTTP notification server that receives and processes upcoming alarms. (see additional “Tixi HTTP Data Interface” manual)
URLPort	TCP/IP port of the HTTP notification server.

For each receiver of your messages you have to prepare the addresses of the transports by which you want to send messages to him.

3.5 Configure Internet Access (ISP)

Tixi Alarm Modem provides the ability to send e-mails via the Internet. For this Tixi Alarm Modem 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 Tixi Alarm Modem and the dial-in node of the ISP. When the TCP/IP connection is ready Tixi Alarm Modem uses the SMTP server of the provider to send e-mail. The related ISP data has to be configured in the 'ISP' database in the 'ISP' section.

You can use the predefined ISP configuration provided by Tixi. In this case select the configuration for your preferred ISP and insert the account data given from the provider:

Database path: /ISP/ISP

```
<ISP>
   <PPPComm>
      <PPPUserName _="user"/>
      <PPPPassword _="pass"/>
      <AuthentFlags _="3"/>
      <FirstDNSAddr _="194.25.2.129"/>
      <SecondDNSAddr _="193.158.131.19"/>
   </PPPComm>
   <SMTP>
      <Flags _="ESMTP"/>
      <mailserver_name _="domain.com"/>
      <mailserver_ip _="192.168.0.1"/>
      <Username _="user"/>
      <Password _="pass"/>
   </SMTP>
   <POP3>
      <mailserver_name _="domain.com"/>
      <Username _="user"/>
      <Password _="pass"/>
      <Flags _="DontDelete"/>
      <Filter _="string"/>
      <Lines _="50"/>
   </POP3>
   <Modem>
      <RemotePhoneNumber _="+49-30-1234567"/>
      <MediaType _="DATA"/>
      <ModemProtocol _="syncPPP"/>
   </Modem>
</ISP>
```

Insert your own data

Name	*Description*
PPPUserName	User name of the PPP log-in (provided by the ISP).
PPPPassword	Password of the PPP log-in (provided by the ISP).
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)

SMTP/Flags	Enter value “POPBeforeSMTP" if you need POP3-before-SMTP authentication. Enter value “ESMTP” if you need SMTP authentication. (@FW 2.0)
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.
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)
POP3/Flags	Enter value “DontDelete" to prevent deleting mails at ISP.
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 modem will skip it. (@FW 2.2)
Lines	Number of lines the modem will search for the filter word.
RemotePhoneNumber	International phone number to call the ISP's dial-in node. 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 Tixi Alarm Modem GSM. The AreaCode will be the network code of your GSM provider, e.g. German ISP T-Online with T-Mobile D1: +49-171-4122
ModemProtocol	ISDN/GSM-Protocol used to connect to the ISP. Values: “default" (uses PSTN or X.75-NL) “X.75-NL" “syncPPP" “V.120" “V.110” “X.75-T.70" “ML-PPP" “HDLC-Transp" “BYTE-Transp"

3.6 Configure the Message Text Template

If it is clear what the event message should say you can define a template for the message text. Typically you can use this template for the SMTP, Fax, Internet e-mail and Express E-Mail. The template itself is an attribute group. The name of the attribute group is identical to the name of the template. The group contains instructions for the Job Generator which creates a text 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.

Elements:

TemplateName :

Name of the template.

Instruction List:

List of instructions to be processed by the template processor. See the following chapter for info on these instructions.

Example: Template which produces the message text, assumed the `Barn` and `Temperature` parameters are given in the `DoOn` command:

Barn is out of temperature

Barn: 12
Temperature of Barn = 10 C

Check the barn.

```
<UserTemplates>
   <AlarmMsg_1>
      <L _="Barn is out of temperature"/>
      <L _=""/>
      <E _="Barn: &#xae;~/Barn;"/>
      <E _="Temperature of Barn: &#xae;~/Temperature; C"/>
      <L _=""/>
      <S _=”Next step:”/>
      <S _="&#xae;~/ToDo"/>
   </AlarmMsg_1>
</UserTemplates>
```

The template processor uses the following instructions

Write a Line

Syntax:

```
<L _="RefText"/>
```

Description:

Writes a text string with a Carriage Return/Line Feed - pair at the end of the text. The text can contain references to other attributes. These references are replaced by the values of the attributes.

Elements:

RefText:

Text to write. The text could include some references to parameters which are placed into the message job templates (see later) or 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;**' character and ends with the ";" 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.

Client event message:

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

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

Lines in the message:
```
    Temperature of Barn: 10 C
    Barn: 12
```

Note: For SMS messages a single line body is used. The line is defined by the message job template as subject (see Message Job Templates chapter 3.7).
If you want to include logfiles into the message text, please see chapter 4.6.

Write a Line – continue on error

Syntax:
```
    <E _="RefText"/>
```

Description:

Writes a text string with a Carriage Return/Line Feed - pair at the end of the text. The text can contain references to other attributes. These references are replaced by the values of the attributes. If Tixi Alarm Modem can't resolve the attribute, it will continue processing the next line.

Elements:

RefText:

Text to write. The text could include some references to parameter which are placed into the message job templates (see later) or 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;**' character and ends with the ";" 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. Due to a missing attribute, Tixi Alarm Modem can't resolve the parameter.

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

Template processor instruction:
```
    <E _="Temperature of Barn: &#xae;~/Temperature; 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 Line – no CRLF

Syntax:
```
    <S _="RefText"/>
```

Description:

Writes a text string without a Carriage Return/Line Feed - pair at the end of the text. The text can contain references to other attributes. These references are replaced by the values of the attributes.

Elements:

RefText:

Text to write. The text could include some references to parameter which are placed into the message job templates (see later) or 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;**' character and ends with the ";" or the end of the tag.

Example:

Writes the line 'Temperature of Barn: 10C Barn 12'. The values are inserted from the client message data attribute barn and temperature.

Client event message:

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

Template processor instruction:

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

Line in the message:

```
     Temperature of Barn: 10C Barn 12
```

Write a Line – no CRLF and continue on error (@FW 2.0)

Syntax:

```
   <C _="RefText"/>
```

Description:

Writes a text string without a Carriage Return/Line Feed - pair at the end of the text. The text can contain references to other attributes. These references are replaced by the values of the attributes. If Tixi Alarm Modem can't resolve the attribute, it will continue processing the next line.

Elements:

RefText:

Text to write. The text could include some references to parameter which are placed into the message job templates (see later) or 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;**' character and ends with the ";" 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. Due to a missing attribute, Tixi Alarm Modem can't resolve the parameter.

Client event message:

```
   [<DoOn _="TemperatureAlert">
      <Temperature _="10"/>
   </DoOn>]
```

Template processor instruction:

```
   <C _="Temperature of Barn: &#xae;~/Temperature;C "/>
   <C _="Barn: &#xae;~/Barn;"/>
```

Line in the message:

```
     Temperature of Barn: 10C
```

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

Include

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 “Signature" into the message text

Signature template:

```
   <Signature>
      <L _="Location:"/>
      <L _="Tixi.Com GmbH"/>
      <L _="Berlin"/>
   </Signature>
```

Text Template with reference to signature:

```
   <Text>
      <L _="Enter your message text here"/>
      <L _=""/>
      <Include _="/D/UserTemplates/Signature"/>
   </Text>
```

InludeLog

Syntax:

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

Description:

See chapter 4.9 “Data logging" for complete reference and examples.

IncludeLogTXT

Syntax:

<IncludeLogTXT _="*LogFileName*" range="*entryrange*" *Formats*/>

Description:

See chapter 4.9 “Data logging" for complete reference and examples.

CopyDatabase

Syntax:

<CopyDatabase _="Path to database"/>

Description:

Copies the defined XML database into the message text.

Elements:

Path to database:

Path to the XML database to be copied into message text

Example:
Copies the EventHandler database into the message text:.

```
   <DatabaseText>
      <L _="EventHandler Database:"/>
      <L _=""/>
      <CopyDatabase _="/EVENTS/D/EventHandler"/>
   </DatabaseText>
```

3.7 Configure Message Job Templates

Typically the reaction to a client event is the sending one or more messages by Tixi Alarm Modem. Each of these is defined by a message job template which defines the message sent 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 the print jobs of network printers.

These templates are defined in the 'TEMPLATE' database. Each template is an attribute group which has a unique name. Insert this name as parameter of the 'SendMail' command in the event handler configuration. The owned attribute defines the predefined template for the creation of a job for the used message transport type like SMTP, SMS, TextFax, Express-E-Mail or HTTP.

The attributes define the default data of this job which are inserted during the creation of the job. Here you can define the receiver and sender address, the template for the message body and the subject line etc.

We recommend to use not more than 100 message job templates.

Templates to create message jobs for a certain client event:

Database path: /TEMPLATE/MessageJobTemplates

```
<MessageJobTemplates>
<TAlertSMTP _="SMTP">
   <Recipient _="/D/AddressBook/TaskForce1"/>
   <Sender _="/D/AddressBook/MySelf"/>
   <Body _="/UserTemplates/AlertMsg_1"/>
   <Subject _="Barn is out of temperature"/>
   <ToDo _="Check the barn"/>
</TAlertSMTP>

<TAlertSMS _="SMS">
   <Recipient _="/D/AddressBook/Addr1"/>
   <Subject _="Barn &#xae;~/Barn; is out of temperature.
   Temp=&#xae;~/Temperature; C."/>
   <Sender _="/D/AddressBook/MySelf"/>
   </TAlertSMS>
</MessageJobTemplates>

<TAlertSMS2 _="GSMSMS">
   <Recipient _="/D/AddressBook/Contact_0"/>
   <Subject _="" Path="/UserTemplates/Alarm1/Subject"/>
   <Sender _="/D/AddressBook/MySelf"/>
   </TAlertSMS2>
</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:

Name of the predefined message job templates for the transport used:

SMTP	predefined template creating a SMTP message job.
SMS	predefined template creating a SMS message job via landline.
TextFax	predefined template creating a FAX message job.
Express-Email	predefined template creating an Express-Mail message job.
GSMSMS	predefined template for sending an SMS message via a GSM modem.
CityRuf	predefined template creating a cityruf - message.
URLSend	predefined template creating a HTTP notification (@FW 2.0)

Example:

Message job template creates an SMTP message job based on the predefined 'SMTP' job template for the 'Temperature Alert' client event.

```
<TAlertSMTP _="SMTP">
   <Recipient _="/D/AddressBook/TaskForce1"/>
   <Sender _="/D/AddressBook/MySelf"/>
   <Body _="/UserTemplates/AlertMsg_1"/>
   <Subject _="Barn is out of temperature"/>
   <ToDo _="Check the barn"/>
</TAlertSMTP>
```

Note: A 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.

For different predefined templates you must insert certain variables as an attribute list in the message job templates. Depending on the transport you must insert the sender and receiver address, the template for the body, the subject etc.

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

Direct (only valid for this MJT, limited to 385 characters):
Subject is written directly into MJT:

```
   <Subject _="Barn is out of temperature"/>
```

Reference (useable for different MJTs, limited to 385 characters):
Subject is written into UserTemplates and referenced within MJT by `&#xae;`:

```
   <Subject _="&#xae;/D/UserTemplates/Message_0/Subject;"/>
```

Referred UserTemplate:

```
   <Message_0>
    <Subject _="Test"/>
   </Message_0>
```

Path (useable for different MJTs, no character limitation):
Subject is written into Usertemplates and included via “Path”:

```
   <Subject _="" path="/D/UserTemplates/Message_0/Subject"/>
```

Referred UserTemplate:

```
   <Message_1>
    <Subject>
      <S _="385 characters text"/>
      <S _="add 385 characters text"/>
      <S _="add 385 characters text"/>
      …
    </Subject>
   </Message_1>
```

Text within `_=""` will be added in front of the UserTemplates text.

SMTP

Name	***Description***
Recipient	Path to the address book entry including the e-mail 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 e-mail 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. So you can use one of these message text templates for separate message job templates.

TextFax

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

Express-Email

Name	*Description*
Recipient	Path to the address book entry including the Express E-Mail Address for the receiver of the Express E-Mail message (inserted in the 'To' field of the message).
Sender	Path to the address book entry including Express E-Mail Address for the sender (inserted in the 'From' field of the message).
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	Text line defining the SMS message text, (max. 160 chars).
Body	not used

CityRuf

Name	*Description*
Recipient	Path to the address book entry including the pager number of the receiver of the SMS message.
Sender	not used.
Subject	Text line defining the message text.
Body	not used

URLSend (@FW 2.0)

Name	*Description*
Recipient	Path to the address book entry including the URL of the receiver of the HTTP notification.
Sender	not used.
Alarm	Alarm number expected by the HTTP notification server.
Secret	Password to access the HTTP notification server.
Parameters	The notification may include parameters which are processed by the HTTP notification server. The parameters field contains a path to the message text template where the parameter is defined. So you can use one of these message text templates for separate message job templates.
Body	See SMTP

For more informations about HTTP notifivations, see additional “Tixi HTTP Data Interface” manual.

3.8 Configure Event Handler

By now you will have defined the events. The next step is to define what Tixi Alarm Modem should do when the event occurs and the event message is received.

The general characteristics are defined in the '**EVENTS**' database. The contents of this database configures the event handler of Tixi Alarm Modem. 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 to use not more than 100 events.

The following example shows the configuration for two events 'TemperatureAlert' and 'FeedAlert', each has its own commands:

Database path: EVENTS/EventHandler

```
<EventHandler>

<TemperatureAlert>
   <SendMail _="MessageJobTemplates/TAlertSMTP"/>
   <SendMail _="MessageJobTemplates/TAlertSMS"/>
</TemperatureAlert>

<FeedAlert>
   <SendMail _="MessageJobTemplates/FAlertSMS"/>
</FeedAlert>

</EventHandler>
```

Event Handler Group

Event Handler Configuration

Event Handler Configuration

Syntax:

```
   <EventName>
      CommandList
   </EventName>
```

Description:

Attribute group which defines the general characteristics of the event handler each time an event message with the same name is received.

Elements:

EventName:

Name of the event sent by the event message from the client or produced by the process subsystem (see chapter 2.4.9.1 on the event parameters).

CommandList:

List of Attributes describing commands for the event handler which are processed from top to bottom.

Example:

Event handler configuration for the 'TemperatureAlert' event. It lets the event handler send an SMTP and an SMS message, defined by the message job templates 'TAlertSMTP' and 'TAlertSMS' respectively.

```
   <TemperatureAlert>
      <SendMail _="MessageJobTemplates/TAlertSMTP"/>
      <SendMail _="MessageJobTemplates/TAlertSMS"/>
   </TemperatureAlert>
```

The attributes of the attribute group describe event handler commands which are commands the event handler processes when the event message is received.

Event Handler

```
<TemperatureAlert>
<SendMail _="MessageJobTemplates/TAlertSMTP"/>
<SendMail _="MessageJobTemplates/TAlertSMS"/>
</TemperatureAlert>

<FeedAlert>
<SendMail _="MessageJobTemplates/FAlertSMS"/>
</FeedAlert>
```

Event Handler Command

3.8.1 EventHandler Commands

SendMail Command

Syntax:

```
<SendMail _="Template">
   <OnOK _="OnOKEvent"/>
   <OnError _="OnErrorEvent"/>
   <MaxRepeat _="MaxRepetitions"/>
   <Interval _="IntervalTime"/>
   <ConfirmID _="ID"/>
   <Timeout _="TimeOut"/>
   <OnTimeout _="OnTimeoutEvent"/>
   <Delay _="DelayTime"/>
   <Priority _="2"/>
</SendMail>
```

Description:

Event handler command. It lets the event handler send a message using the given message job template.

Elements:

Template

Name of the message job template which is used to generate the message job for this event (see message job templates).

OnOKEvent

Name of the event to be triggered when the sending of the ***Template*** message job didn't fail. 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. (Requires *Interval*)

IntervalTime

Determines the delay in seconds 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 (in milliseconds) the OnTimeout event is invoked. Timer starts with begin of sendmail.

OnTimeout

Name of the event to be triggered when the message could not be sent or no confirmation was received after the time determined in ***Timeout***. If empty or omitted, nothing happens in that case.

DelayTime (@FW 2.0)

Delays the sending of the messsage for the given time. Timer starts with created message.

Priority

Sets a priority for an alarm. If several alarms are activated at the same time, the device will send out the alarm with the highest priority (2 is higher than 1) at first. Possible Priorities: 1-255

Example:

Event handler configuration for the 'TemperatureAlert' event. It lets the event handler send an SMTP and an SMS message using the templates 'TAlertSMTP' and 'TAlertSMS' respectively.

```
<TemperatureAlert>
  <SendMail _="MessageJobTemplates/TAlertSMTP"/>
  <SendMail _="MessageJobTemplates/TAlertSMS"/>
</TemperatureAlert>
```

Alarmcascading 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 to combine OnTimeout with OnError.
- If the OnTimeout timeout is shorter than the time for all MaxRepeat intervals, OnTimeoit 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`*`"/>.`** The ID (0-65533) is an identifier for the sent message. The ConfirmID has to be included in the message in form of a fingerprint.

In this message template the fingerprint and the ConfirmID (for information) is included in the subject of the message.

Database path: /TEMPLATE/MessageJobTemplates

```
<MessageJobTemplates>
   <SMSAlarmMsg _="GSMSMS">
      <Recipient _="/D/AddressBook/Receiver"/>
      <Sender _="/D/AddressBook/Myself"/>
      <Subject _="Alarm! confirmation needed:  &#xae;~/_Fingerprint;
      (&#xae;~/_ConfirmID;)"/>
   </SMSAlarmMsg >
</MessageJobTemplates>
```

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

Database path: /EVENTS/EventHandler

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

The message would look similar to this:

```
Alarm! confirmation needed: CID2VeFhc7SyfaJMT/h (100)
```

The confirmation is performed by a Confirm command in an event handler. There are two ways to invoke this event handler:

1. As a reply message with the received subject copied into the replying one. The Tixi Alarm Modem searches the text of received SMS and the subject lines of Email or Express-E-Mail 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 Tixi Alarm Modem, it invokes a special system event /System/Confirmation (necessary) which has to contain a confirm command and may contain additional commands (such as logging or switching via set command).

Database path: /EVENTS/EventHandler/System

```
<EventHandler>
   <System>
     <Confirmation>
         <Confirm _="®~/_ConfirmID"/>
         <Log _="EventLog">
              <ConfirmID _="®~/_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, over a DoOn command (chapter 2.4.9.1), as an event in a command message (chapters 8), or via a port change or service button (chapter 6).

Example:

The service button has to be pressed by the service personal to log the time of arrival after an alert.

Database path: /EVENTS/EventHandler/System

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

ConfirmID of EventHandler

***Set** - Set System Properties*

Syntax:

```
<Set _="Path" value="Value"/>
```

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 Tixi Alarm Modem. 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 UserTemplates or the EventHandlers which could contain some elements with the same name. 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.
The value may be created by references. The value string is limited to 80 characters.

Example:

Set the relais output of a hutline modem.

```
<SetRelais>
   <Set _="/Process/MB/IO/Q/P2" value="1"/>
<SetRelais>
```

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 where the log entry is written.

Note: The Log command can only be used for logfiles defined with the content type **Note:** “binary".

Elements:

LogData:

XML formatted data to be logged.

LogfileName:

Name of the logfile to be used. Must be defined in the LogFiles database.

Note: If attributes are used like `<PortWindowOpen _="&#xae;/Process/MB/IO/I/P4"/>` you can insert references to any system property. The reference used will then be replaced by the corresponding value.

<u>Example</u>: Event handler logs the last power off and the last power on time.

```
<PowerOn >
   <Log>
      <PowerOff _="&#xae;/TIMES/PowerOffTime;"/>
      <PowerOn _="&#xae;/TIMES/PowerOnTime;"/>
   </Log>
</PowerOn>
```

BinLog Command

<u>Syntax:</u>

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

<u>Description:</u>

Creates an 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.

<u>Elements:</u>

LogfileName:

Name of the logfile to be used. Must be defined in the LogFiles database.

ValueName:

Name of value defined in record database

Value:

Value to be written into the structure.

Process Command

<u>Syntax:</u>

```
<Process>
   Instruction List
</Process>
```

<u>Description:</u>

Processes the instructions of the given instruction list.

<u>Elements:</u>

Instruction List:

List of instructions calculating the value of the process variable (for a description of instructions see Configuring Process Variables chapter 6.2).

Note:

To process a variable given by the event parameter you can use the data path (~/) as address parameter of the instruction:
Example:

Assume the event command:

```
<DoOn _="SetPort1">
   <PortValue _="1"/>
</DoOn>
```

The Event handler sets the port P1 to the value given by the event command.

```
<SetPort1>
   <Process>
      <LD _="&#xae;~/PortValue"/>
      <ST _="MB/IO/Q/P1"/>
   </Process>
</SetPort1>
```

Example: Event handler sets the output port P4, logs this port and sends an SMS.

```
<SetPort4>
   <Process>
      <LD _="1"/>
      <ST _="MB/IO/Q/P4"/>
   </Process>
   <Log>
      <Action _="Port set"/>
      <Portstate _="&#xae;/Process/MB/IO/Q/P4;"/>
   </Log>
   <SendMail _="MessageJobTemplates/ConfirmSMS"/>
</SetPort4>
```

Note: Even the `Set` command can be used inside event handler configuration. This command is described in detail in chapter 2.4.8.4 of this manual.

Delay Command

Syntax:

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

Description:

Includes a delay between two instructions.

Elements:

Instruction:
EventHandler command, e.g. “SendMail" or “Set".

X: Time in seconds (1-60)

Example:

Event handler sets the PLC variable 1, waits 5 seconds and processes thereafter the SendMail.

```
<SMSSetPort4>
   <Set _="/Process/Bus1/Device_0/Variable_1" value="1"/>
   <Delay _="5s"/>
   <SendMail _="MessageJobTemplates/ConfirmSMS"/>
</SetPort4>
```

This may be usefull if a “Set" of a PLC variable is initiated by an incoming SMS, 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 also SendMail parameter “OnTimeout”).

Elements:

ConfirmID:
ID given in SendMail command or “*” (@FW 2.2) to confirm all jobs.

Example: This EventHandler confirms a SendMail command waiting for acknowledge. The requested ConfirmID was 99.

```
<ConfirmMessage>
   <Confirm _="99"/>
</ConfirmMessage>
```

The Confirm command is in most cases used by the system events “OnButton” (see 6.8) and “Confirmation” (see SendMail command).

SetConfig Command (@FW 2.0)

Syntax:

```
<SetConfig/>
```

Description:

EventHandler command to change databases via incoming Email or Express-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 Tixi Alarm Modem.
For further information read chapter 9.6.

Elements:

No elements

Example:

A incoming message with subject “Password LoadDatabase” (Password: see chapter 9.7) and a database included in message body will be processed by this EvantHandler:

```
<LoadDatabase>
   <SetConfig/>
</LoadDatabase>
```

POP3Query Command

Syntax:

```
<POP3Query/>
```

Description:

Queries a configured POP3 account (see chapter 3.5) for new emails to process incoming messages (see chapter 9.4.3.3).

Elements:

No elements

Example: This EventHandler may be called periodically via scheduler to query a POP3 account for new emails to process:

```
<GetMails>
   <POP3Query/>
</GetMails>
```

Clear Command (@FW 2.0)

Syntax:

```
<Clear Log=”Logfiles”/>
```

Description:

Deletes the content of one or several logfiles. (see chapter 2.4.9.2).

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.

Example: This EventHandler may be called on OnOK cascading after the logfile was sent via email:

```
<ClearLog>
   <Clear Log=”Datalog”/>
</ClearLog>
```

Reset Command (@FW 2.0)

Syntax:

```
<Reset/>
```

Description:

Processes a “Reset keep” of the modem (see chapter 2.4.7.2).
Reset will be executed with a delay of 10s.

Elements:

No elements

Example: This EventHandler may be called periodically via scheduler to reset the modem.

```
<ResetModem>
   <Reset/>
</ResetModem>
```

INetTime Command

Syntax:

```
<InetTime/>
```

Description:

Queries a Internet TIME-Server to synchronize the RealTimeClock of the device.
See chapter 3.12 for more informations.

Elements:

No elements

Example: This EventHandler may be called once a month via scheduler to synchronize the Alarm Modems clock with a Internet TIME-Server:

```
<TimeSync>
   <INetTime/>
</TimeSync>
```

SetTime Command (@FW 2.0)

Syntax:

```
<SetTime _="Time" TimeDiff="Difference"/>
```

Description:

Sets the Alarm Modem clock (RTC) to the given value. May be used to synchronize the Alarm Modem 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 TimeZone of the /USER/USER database will be added to the Difference.
+/-HHMM

Example: This EventHandler copies the value of the PLC time variable into the Alarm Modem RTC and adds one hour (/USER/USER/Timezone=”+0000”):

```
<CopyTime>
 <SetTime _="&#xae;/Process/Bus1/Device_0/Clock;" TimeDiff="+0100"/>
</CopyTime>
```

CBIS Command (@FW 2.0)

Syntax:

<CBIS/>

Description:

Connects the Alarm Modem to the Internet and sends its IP address to a predefined Email address. (see Webserver TiXML Manual)

Elements:

No elements

Example: This EventHandler starts the CBIS procedure:

```
<CallBack>
   <CBIS/>
</CallBack>
```

S0_Sync Command (@FW 2.0)

Syntax:

<S0_Sync/>

Description:

This command is only used together with the S0-interface (see chapter 0). It generates a synchronization impulse by the modem (e.g. via scheduler) instead of using an external synchonization impulse. The created synchronization impulse copies the counted S0 impulses into the counter variable.

Elements:

No elements

Example: This EventHandler creates an synchronization impulse:

```
<SyncImpulse>
   <S0_Sync/>
</SyncImpulse >
```

Switch Command (@FW 2.2)

Syntax:

<Switch _="*n*"/>

Description:

This command sets the mode of the device.

Elements:

n:

ModemMode Tixi Alarm Modem responds to the AT command set.
TiXMLMode Tixi Alarm Modem responds to the TiXML.

Example: This EventHandler switches the Alarm Modem into Modem Mode

```
<SwitchMode>
   <Switch _="ModemMode"/>
</SwitchMode >
```

3.8.2 Event "IF" condition

The IF instruction is used to enable/disable event processing in special conditions. E.g. it may be used to deactivate scheduled data logging during transmode or any other necessary condition.

If instruction (@FW 2.2)

Syntax:

```
      <If _="Condition">
         EventHandler commands
      </If>
```

Description:

Processes the enclosured EventHandler commands only if the condition is eqauls "1".

Elements:

Condition

Path to a bit variable, e.g. ProcessVar

EventHandler commands:

List of EventHandler commands (for a complete list see chapter 3.8.1).

Example: Scheduled data logging is only processed if PLC communication is active.

```
<Datalog>
  <If _="/Process/Bus1/Device_0/DeviceState">
   <Log _="Port" >
      <PortLog1 _="&#xae;/Process/Bus1/Device_0/Word01"/>
      <PortLog2 _="&#xae;/Process/Bus1/Device_0/Word02"/>
   </Log>
  </If>
</Datalog>
```

3.9 Configure additional SMS provider

The ISP database contains a section SMS_Provider which describes the access to the services of the SMS provider.

Name	***Description***
ProviderName	Name of the provider (Random names may be chosen but must be unique throughout the provider list).
Dialin	Phone number of the SMS provider.
Type	This can be either 'SMS' – Use the PSTN interface (SMS via fixed phone lines) via 1TR140 protocol. 'Script' – Send the SMS via a common modem (TAP, UCP) or GSM connection (this is the standard way to send SMS).
Script	The protocol for the SMS transmission (only necessary if Type=Script). Can be either 'D1_TAP' (8N1), 'Mobilkom_A_TAP' (7E1), 'D2_UCP' or 'GSM'.
NumberFormat (@FW 2.0)	Used to convert an addressbook number (canonical) into the required format, e.g. addressbook: +49-172-1234567 will be send as: "national": 01721234567 "canonical": +491721234567
SMS_ISDN	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.
SMS_Media	Has to be 'SMS' for GSM providers.

Example:
Modem Gateways (UCP, TAP):
The following example configures the access to the SMS Modem-Gateway of the German D2 network:

Database path: /ISP/SMS_Provider

```
<D2_ISDN>
   <Dialin _="+49-172-2278025"/>
   <Type _="Script"/>
   <Script _="D2_UCP"/>
   <NumberFormat _=”national”/>
   <SMS_ISDN _="X.75-T.70"/>
</D2_ISDN>
```

PSTN-Gateways (1TR140):
This example configures the access to the SMS PSTN Gateway of the German provider “AnnyWay”:

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

GSM-SMSC:
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 theire SMS gateways (GSM, TAP, UCP, 1TR140).

3.10 Configure 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 at the AnnyWay service center.

Database path: /ISP/IncomingSMSCenter

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

3.11 Configure access rights

You can protect the access to the Tixi Alarm Modem against unauthorized access. If you do not need this, skip this part. The factory configuration of the Tixi Alarm Modem has no protection activated, so you can work without a knowledge of this protection function when you start becoming acquainted with Tixi Alarm Modem's functions.

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)

3.11.1 Simple access rights

The level of protection is controlled by the configuration of the 'Login' section of the USER database. If **no protection** is required, this section is empty:

```
   <Login>
   </Login>
```

In this case all commands are processed. The Login command returns successfully with every user name and password you will give it (see the meaning of the Login command in this state in the chapter where the command is described).

If a **password protection** is required, a user name password map is set with "Default" as user name and the desired password.

Password

```
   <Login>
   <Default _="secret"/>
   </Login>
```

In this case the user name is empty and is sent together with the right password for the Login command.

Attention: When the password protection is set from no protection state by means of the SetConfig command, all following commands are rejected until a successful login is done.

When you change your password or user name, the previously done login is valid until the Logout command is sent. After this, the new password or username password pair must be used.

When a protection is necessary you can set a map of user name with password in the configuration. In this case, all users identified by their name are authorized to access the system. To check this the user must send the Login command with the user name and the corresponding password. All of these users have the same rights to access the device.

User Password Map

```
<Login>
   <General _="TixiOK"/>
   <Service _="Service"/>
   <OEM _="OEMPwd"/>
</Login>
```

3.11.2 Advanced access rights

With firmware 2.0 a new access rights configuration with different access levels was introduced. Now it is possible to define access groups with different access types 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 (@FW 2.0)

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 expicit 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)
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

OA_nnn: CallerID or sender alias used to secure remote switching (see chapter 9.7)

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 (separated by comma).
Number:	Callback number for TSAdapter service (only Hx71/Hx76)
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 Tixi Alarm Modem but he cannot login to the device.

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 login 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”/>
                   <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” />
          <Paul    Pwd=”Agshezg435G73gg723==”  Group=”Login,RemoteControl” />
          <Def_RemoteLogin Plain=”Winter” Group=”Guest”/>
          <Def_LocalLogin Plain=”Summer” Group=”Guest”/>
          <ADMIN Plain=”Autumn” Group=”Step7” Callback=”+491721234567”/>
        </User>
    <AccRights>
```

3.12 Configure automatic transmode

The Tixi Alarm Modem is able to redirect some (callerID) or all incoming calls to one of its serial interfaces.

This offers TransMode capability without need to send Login or TransMode commands.

Database path: /ISP/AutoTransMod

ISP Database – automatic TransMode (@FW 2.2)

Syntax:

```
<AutoTransMode>
   <No1 _="CID" transmode="comport" format="SerialFormat"
   baud="Baud Rate" handshake="Handshake" wait="timeout"/>
</AutoTransMode>
```

Description:

1. It switches the remote Tixi Alarm Modem to a transparent mode (like Modem Mode). It connects the modem to the selected extension com port or the host port.
2. It transforms the baud rate and the serial data format from the phone line baud rate and format to the values the remote client device uses.

Note: This transparent mode of the remote Tixi Alarm Modem is finished when the dialup connection drops down. After this the remote Tixi Alarm Modem goes back in the TiXMLMode. To drop the connection the local desktop PC must send the escape sequence (+++) and ATH, if the TransMode command was issued over the modem. A transparent mode to the host port MB (COM1) is blocked if a local login session is open (see chapter 2.4.7.3).

Elements:

X:

Increasing number of entries. Value 1 - 2

CID:

CallerID used to trigger the automatic ransmode entry

comport:

Specifies the COM port on the Tixi Alarm Modem used for the connection.

Hutline Modems:

`COM1`	Programming port (labeled ***COM1 RS232***) (**default**)
`COM2`	PLC port (labeled ***COM2 RS232*** or ***COM2 R-485/422***) (if available)

Aluline Modems:

`COM1`	mainboard port (labeled ***RS232(1)***) (**default**)
`COM2`	port on extension board #0 (labeled ***RS422/485)***) (if available)
`COM3`	port on extension board #1 (labeled ***RS232(2)***)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:

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`	disables DTR

Note: RS 485/422 communication is only possible with special RS 485/422 interfaces.

timeout:

Specifies the time the Tixi Alarm Modem will try to disable a PLC bus protocol on the remote com port (Default: 20s).

<u>Example:</u>

Establish Transmode to COM2 with 38400bps, data format 8O1 and hardware handshake if call with callerID 0301234567 is detected and (No2) establish Transmode 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 Transmode 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 Configure Internet-Time synchronization

The Tixi Alarm Modem uses a battery buffered Real Time Clock. Add following configuration to synchronize the Clock with an Internet Time Server via Time / DayTime Protocol (TCP Port 13 / 37):

The time server,used protocol and time difference settings are located in ISP database, ISP group.

Database path: /ISP/ISP/TimeServer

ISP Database – Internet Time synchronization

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 (only DAYTIME).

Elements:

address

Address of the internet time server. “time.nist.gov“ is the predefined time server.

protocol

Protocol used to query the time server.

DAYTIME:	DayTime protocol on port 13 (predefined protocol for “time.nist.gov“)
TIME:	Time protocol on port 37

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 modem 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:

Default time server settings:

```
<TimeServer>
   <ServerName _="time.nist.gov"/>
   <Protocol _="DAYTIME"/>
   <TimeDiff _="+0000"/>
</TimeServer>
```

German atom clock DAYTIME server settings:

```
<TimeServer>
   <ServerName _="ptbtime2.ptb.de"/>
   <Protocol _="DAYTIME"/>
   <TimeDiff _="-0100"/>
   <TimeFormat _="d E y h:n:s "/>
</TimeServer>
```

To synchronize the Alarm Modem Time with the time server a simple EventHandler will do the job:

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

A good solution can be implemented by combining the Time query event with the scheduler. This will synchronize the time 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 winter/summer time calculation):

```
<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>
```

3.14 Configure Ethernet Module

The Tixi Alarm Modem may be upgraded by an ethernet module for HTTP- and TiXML-access.

The necessary TCP/IP configuration is located in the ISP database, Ethernet group.

Database path: /ISP/Ethernet

ISP Database – Ethernet configuration (@FW 2.2)

Syntax:

```
<Ethernet>
   <IP _="IP-address"/>
   <Mask _="Subnetmask"/>
   <Gateway _="GW-address"/>
   <FirstDNSAddr _="DNS-address"/>
   <SecondDNSAddr _="DNS-address"/>
</Ethernet>
```

Description:

Defines the TCP/IP settings of the Tixi Alarm Modem Ethernet module.

Elements:

IP-address

Static IP address of the Tixi Alarm Modem Ethernet module in “dotted quad format”.

Subnetmask

Subnetmask according to the IP address of the Tixi Alarm Modem Ethernet module.

GW-address

Gateway IP address of the next router.

DNS-address

IP address of the DNS server.

Example:

Private CLASS-C network with a WAN router.

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

3.15 Testing

After configuring the mentioned properties, you may run a test with Tixi Alarm Modem.

1. **Create an event message:**
   ```
   [<DoOn _"TemperatureAlert">
      <Barn _="12"/>
      <Temperature _="10"/>
   </DoOn>]
   ```
2. **Send this command to Tixi Alarm Modem (Process LED goes on).**
3. **Check the result of this command.**
   If not OK a job could not be created. In this case check the templates used.
4. **Check the resulting activities of Tixi Alarm Modem:** Mail Out LED goes on, Line LED goes on, Mail Out LED goes off.

5. **Check the results of the message jobs by sending (see Chapter 5, Logging)**
   `[<ReadLog _="LogFileName" range="#1-"/>]`
   Note that you need to insert the actual logfile names.
6. **Check the Logfile content for the notifications on your messages.**
   To check the results look for the attribute value of ErrNo. If the attribute ErrNo = 0 the message is sent without error. If not,, look for the error description and the additional state data about the error.

```
<ReadLog>
    <ID_1 _="2001/09/05,13:44:07">
    <JobReport>
        <ID _="3"/>
        <Time _="2001/09/05,13:43:56"/>
        <Type _="TextFax"/>
        <JTState>
            <ErrNo _="-300"/>
            <ErrText _="modem connection failed"/>
            <Line _="505"/>
            <Module _="ModmComS"/>
            <Class _="TXModemCommService"/>
            <StartTime _="2001/09/05,13:43:57"/>
            <EndTime _="2001/09/05,13:44:07"/>
            <Attempts _="1"/>
        </JTState>
        <Fax>
            <SConState _="stIdle"/>
        </Fax>
        <Modem>
            <SConState _="stError"/>
            <ModemResult _="NoCarrier"/>
            <DialString _="ATX3DT00040608400"/>
        </Modem>
        <FaxTransmission>
        </FaxTransmission>
    </JobReport>
    </ID_1>
</ReadLog>
```

4 Data Logging

Any operation of Tixi Alarm Modem may be logged for later review of what actually happened. Different kinds of incidents can be monitored, and a maximum of 12 logfiles can be created in order not to store all info in the same place.

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

By default, no logfiles are created. The LOG database is therefore empty in the factory state and every logfile to be used must be stated explicitly.

The logfiles themselves are organized as ring buffers with a user defined size. If a logfile is full, then the logging starts at the beginning of the logfile overwriting the oldest logfile entries.

Two different types of logfiles are supported:

- XML-logfiles for xml-formatted logging (easy to read)
- Binary logfiles for BASE64 coded logging (less memory usage)

4.1 The LogFiles Database

Database path: /LOG/Logfiles

LogFiles Database

Syntax:

```
<LogFileName size="LogFileSize"contenttype="Type"
   record="RecordPath"/>
```

Description:

Defines logfiles by their name and size.

Elements:

LogFileName

The name identifier of the logfile. Random names may be chosen but must be unique throughout the configuration. A maximum of 12 logfiles can be created.

LogFileSize

Specifies the logfile size in bytes. In order to delete a logfile, set this to zero. 1024, for example, creates a 1KB logfile. File size bigger than Tixi Alarm Modem memory will be rejected with an error message. Because of the file system structure the logfile size should be divideable by 512.

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 log entry.

Example:

Create XML-logfile *Log1* with 1KB size and binary logfile *Log2* with 20KB size and 80 bytes maximum for an entry:

```
[<SetConfig _="LOG" ver="v">
   <LogFiles>
      <Log1 size="1024"/>
      <Log2 size="20480" contenttype="binary" record="Struct"/>
   </LogFiles>
</SetConfig>]
```

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 is necessary to do a factory reset before uploading the new logfile definition.

4.1.1 SupportLog

After a factory reset the Tixi Alarm Modem is preconfigured with a Logfile “SupportLog” which may also be added to TiXML projects. (@FW 2.2)
This logfile is used to collect information about the incorporated PSTN or GSM modem like country setting, SMSC, own numbers etc. during system startup.

4.2 The Records Database

Database path: /LOG/Records

Records Database

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"/>
</RecordName>
```

Description:

Defines the structure of an binary log entry.

Elements:

RecordName

Name of the data structure the logfile refers to.

ValueName

Name of the value beeing 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 6.7.1 (@FW 2.2)

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). Has to be defined for type “string”, optional for all other types except “meterbus” (variable size)

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` (@FW 2.0)

Exponent of base 10 to specify fix point precision of
simpleType = Uint8, Uint16, Uint32, Int8, Int16, Int32 (see 6.7.1).
The logged variable value will be multiplied by 10 exp(*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 (@FW 2.0)

The logged value will be multiplied by this factor to get the output value.
simpleType = Uint8, Uint16, Uint32, Int8, Int16, Int32 (see 6.7.1).

Output value = Factor * Logged value

The factor is used a a fraction, e.g.: „1/1000“ or „3600/1“, the denominator and numerator must not be zero.

Source (@FW 2.2)

Source for the log entry value, if not specified in the log command (option). Works faster than “value” but supports external (PLC) variables only.

FormatString

String that defines the logfile value output format.

For a list of available format option see chapter 6.7.

Example:

This example creates a data structure with 3 values:

```
[<SetConfig _="LOG" ver="yes">
<Records>
   <Struct>
      <Value1 _="int" size="2"/>
      <Value2 _="int" size="3"/>
      <Value3 _="int" size="1" value="&#xae;/Process/C40/IB/P0;"/>
      <Value4 _="int" size="1" value="&#xae;/Process/C40/I/P8;"
       format="?On,Off;"/>
      <Value5 _="byte"/>
      <Value6 _="byte" path="/Process/C40/QB/P0"/>
   </Struct>
</Records>
</SetConfig>]
```

Value1 may be a Int16 signed 16bit word value (2 byte) with a value range of -32768 to 32767.

Value2 may be a Uint16 unsigned 16bit word value (3 byte) with a value range of 0 to 65535.

Value3 is a 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.

Value4 is a 8bit number value (1 byte) with a value range of -128 to 127. The input port P8 is automatically written as this byte. Instead of the 0/1 value the string “On" or “Off" will be omitted during reading the logfile.

Value5 may be a Uint8 unsigned byte value with a value range of 0 to 255.

Value6 may be a Uint8 unsigned byte value with a value range of 0 to 255. Value assigned via “path”.

4.3 New LogDefinition database

To prevent upload problems of the LOG database groups (records for logfiles missing - vice versa) Tixi.Com has made the decision to redesign the LOG database with introduction of firmware 2.0. We recommend to use the new structure even if the old format is still supported (Note: don't mix old and new structure).
The "LogFiles" and the "Records" group are now both part of the "LogDefinition" group inside LOG database. The structure inside both groups didn't change. Refer to chapters 4.1 and 4.2 for more informations.

Database path: /LOG/LogDefinition

Example:

```
[<SetConfig _="LOG" ver="v">
<LogDefinition>
   <LogFiles>
      <Event size="10240"/>
      <JobReport size="10240"/>
      <DataLog size="20480" contenttype="binary" record="Struct"/>
   </LogFiles>
   <Records>
   <Struct>
         <Value1 _="int" size="2"/>
         <Value2 _="int" size="3"/>
   </Struct>
   </Records>
</LogDefinition>
</SetConfig>]
```

4.4 Record format options

Without any formatting the value of the variables during processing logfiles will be shown as retrieved by the PLC or I/O processor. Tixi Alarm Modem is able to reformat the value into a number format or to replace the status of a boolean variable with a string. The reformatted variable will be used during reading or sending logfiles.

Example:

```
      <Value3 _="int" size="1" value="&#xae;/Process/MB/IO/I/P8;"
         format="?On,Off;"/>
```

Value3 is an boolean variable, but with the format string the logged data will show "On" or "Off" instead of "0" or "1".

Valid format instruction can be found in chapter 6.7.2.

4.5 The EventLogging Database

Database path: /LOG/EventLogging

EventLogging Database
Syntax: **`<EventSource mode="Mode1Mode2" file="LogFileName"/>`**
Description: Defines what info 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 client events that become processed
Login	reports all cases of anyone doing a login into the Tixi Alarm Modem as well as for logout
IncomingMessage	reports on all incoming messages
FailedIncomingCall	reports on all incoming calls that could not be handled properly
JobReport	reports the result of sending a message, regardless if it was ok or error.

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. One of the names may be chosen that were configured in the LogFiles database. There is a maximum of 13 logfiles to be created.

Example:

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

```
[<SetConfig _="LOG" ver="v">
   <EventLogging>
      <JobReport mode="ev" file="Log1"/>
      <Event mode="a" file="Log2"/>
   </EventLogging>
</SetConfig>]
```

4.6 Logging commands

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 where the log entry is 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 LogFiles database.

Note: If attributes are used like
`<PortWindowOpen _="&#xae;/Process/MB/IO/I/P4"/>`
you can insert references to any system property. The reference used will then be replaced by the corresponding value.

Example: Event handler logs the last power off and the last power on time.

```
   <PowerOn >
      <Log _="Log1">
         <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 an 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 LogFiles database.

ValueName:
Name of value defined in record database

Value:
Value to be written into the structure.

Example: This example uses the logfile and record definitions shown above. Value1 is given as “Parameter" during DoOn. Value2 and Value3 are port values given by the record value definition.

```
<LogValues>
   <BinLog _="Log2">
      <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>
```

4.7 Calculating logfile memory

To calculate the necessary memory and logfile size these informations are usefull:

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:

```
<Struct>
   <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“/>
</Struct>
```

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

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

4.8 Reading and clearing logfiles

For informations on how to read or clear the content of logfiles read chapter 2.4.9.2.

4.9 Sending and formatting log reports

Tixi Alarm Modem is able to include logged data into messages. Due to the XML databases, the logged data is also stored in XML format. You can choose two different commands to include logdata into messages:

1. **IncludeLog**. The logged data will be included in XML format, just as they are stored in the file system.

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 2.4.9.2 for details.

Parameter:

LogFileName:

Name of the logfile to be read.

entryrange:

Range of log data to send. See chapter 2.4.9.2 for details.

Message Example:

Send the entries with the IDs 7 – 8.

```
<UserTemplates>
   <LogfileMsg>
      <IncludeLog _="Journal" 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" Formats/>
```

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 depends on the specified format parameters .

Parameter:

LogFileName:

Name of the logfile to be read.

entryrange:

Range of log data to send. See chapter 2.4.9.2 for details.

templates:

Predefined logfile formats (@FW 2.0):

CSV:	“comma separated format”, 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" (only for templates CSV/HTML)

NoId:	removes the ID of each entry
NoDate:	removes the Date of each entry
NoTime:	removes the Time of each entry
NoNames:	removes the first row with variable names (@FW 2.2)

Formats:

tabstart tabstart="*TS*"

TS: string which will be added at the beginning of the file
maximum length: 30 chars
default: (empty)

tabend tabend="*TE*"

TE: string which will be added to the end of the file
maximum length: 30 chars
default: (empty)

tagstart tagstart="*ts*"

ts: string which will be added in front of each value.
maximum length: 30 chars
default: “

tagend	tagend="*te*"
te:	string which will be added to the end of each value. maximum length: 30 chars default: “
colsep	colsep="*cs*"
cs:	string which will be added between the values (between tagend and tagstart). maximum length: 30 chars default: ;
rowstart	rowstart="*rs*"
rs:	string which will be added in front of the first value (in front of tagstart). maximum length: 30 chars default: (empty)
rowend	rowend="*re*"
re:	string which will be added to the end of the last value (after tagend). maximum length: 30 chars default: (CRLF)

Attention: If you enter any character as rowend, 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;"
```

<u>Message Examples:</u>

Logged Data:

```
<ID_7 _="2002/10/10,16:10:51">
   <Data1 _="Logged Data1"/>
   <Data2 _="Logged Data2"/>
   <Data3 _="Logged Data3"/>
</ID_7>

<ID_8 _="2002/10/10,16:10:51">
   <Data1 _="Logged Data1"/>
   <Data2 _="Logged Data2"/>
   <Data3 _="Logged Data3"/>
</ID_8>
```

Example 1:

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

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

Message Body result:

```
"Date";"Time";"Data1";"Data2";"Data3"
"2002/10/10";"16:10:51";"Logged Data1";"Logged Data2";"Logged Data3"
"2002/10/10";"16:10:51";"Logged Data1";"Logged Data2";"Logged Data3"
```

Example 2:

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

```
<UserTemplates>
   <LogfileMsg>
      <IncludeLogTXT _="Journal" range="ID_7-ID_8"
      flags="NoId,NoDate,NoTime,NoNames" type="HTML"/>
   </LogfileMsg>
</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 3:

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

```
<UserTemplates>
   <LogfileMsg>
      <IncludeLogTXT _="Journal" range="ID_7-ID_8"
      flags="NoId,NoDate,NoTime,NoNames" tagstart=”#” tagend=”#”
colsep=”+” rowstart=”-“ rowend=”-&#x0a;&#xod;”/>
   </LogfileMsg>
</UserTemplates>
```

Message Body result:

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

4.9.1 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

4.9.2 Sending logfiles as attachment

You can enhance the IncludeLogTXT message text template with special MIME headers. This will separate the message body from the logfile and you can easily use the received email attachment in your data application. The email attachment will be base64 coded.

The MessageJobTemplate has to have a new parameter: “Attachments" (@FW 2.0)

Database path: /TEMPLATE/MessageJobTemplates

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

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

Database path: /TEMPLATE/UserTemplates

```
<AttachmentGroup>
   <Attachment filename="file">
      <Content/>
      <Content/>
   </Attachment>
   <Attachment filename="file">
      <Content/>
      <Content/>
   </Attachment>
…
</AttachmentGroup>
```

Example:

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

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

Note for Firmware <2.1.27.0 (“NoNames” not available):

To add column names in the first line of a CSV attachment, the names could be directly written as a new line with CSV format above the IncludeLogTXT command:

```
< Attachment_CSV >
   <Attachment filename="Journal.csv">
      <L _="Date;Time;Column1;Column2;Column3"/>
      <IncludeLogTXT _="Journal" range="previous 1 days"
   flags="NoId" type="CSV"/>
   </Attachment>
</ Attachment_CSV >
```

To add column names in the first line of a HTML attachment, the names could be added with following syntax:

```
< Attachment_CSV >
   <Attachment filename="Journal.csv">
      <L _="&lt;table&gt; "/>
      <L _="&lt;tr&gt; "/>
      <L _="&#x09;&lt;td&gt; Name1 &lt;/td&gt; "/>
      <L _="&#x09;&lt;td&gt; name1a &lt;/td&gt; "/>
      <L _="&#x09;&lt;td&gt; Name2 &lt;/td&gt; "/>
      <L _="&lt;/tr&gt;“/>
      <IncludeLogTXT _="Journal" range="previous 1 days"
   flags="NoId" tabstart="" type="HTML"/>
   </Attachment>
</ Attachment_CSV >
```

4.10 Logfile Counter

The Alarm Modem automatically creates log counters within system properties path `/LogCounter/Logfilename` (see chapter 12). (@FW 2.0)

- 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 2.4.8.4).
- 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

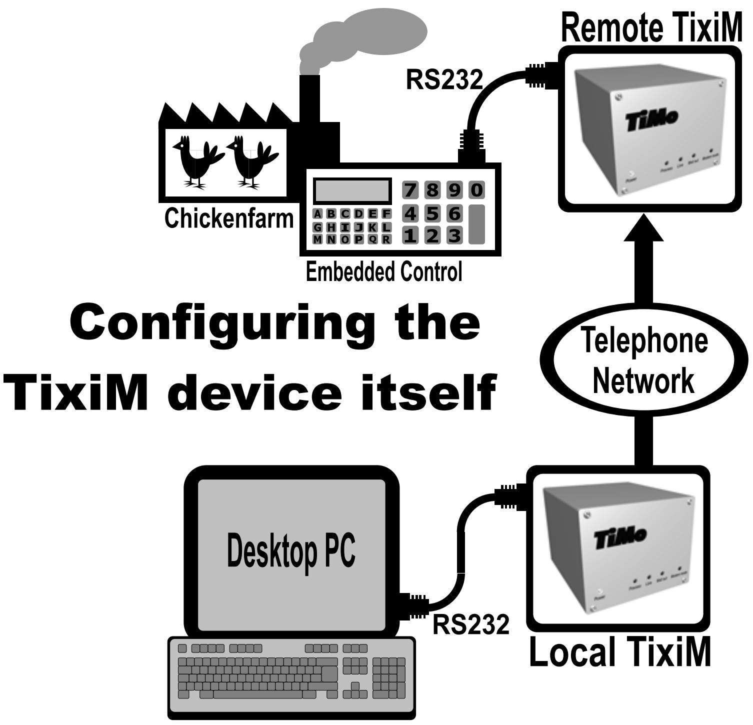
5 Remote Control

5.1 Overview

Using the Tixi Alarm Modem, it is possible to remotely control both the Tixi Alarm Modem and the connected client device. The following chapters describe these two cases in detail. Read the chapter 'Remote Control of the Tixi Alarm Modem' first because the Remote control of the client device is based on this chapter.

5.2 Remote Control of the Tixi Alarm Modem

The following picture shows the configuration when controlling the Tixi Alarm Modem by remote.



In this configuration the desktop PC controls the remote Tixi Alarm Modem by means of the TiXML protocol. To do this the following steps are required:

1. The Desktop PC controls the local Tixi Alarm Modem by means of TiXML. It sends the **Remote command** with the user, password and phone number to Tixi Alarm Modem.
2. The local Tixi Alarm Modem processes the Remote command and dials the phone number to establish a dialup connection to the remote Tixi Alarm Modem. When this is done it makes a login to the remote Tixi Alarm Modem. When the Login is OK the local Tixi Alarm Modem switches to the Modem Mode and there is a transparent serial connection between the desktop PC and the remote Tixi Alarm Modem.
3. The desktop PC controls the remote Tixi Alarm Modem by means of **TiXML**.
4. The remote connection is terminated by the **Logout** command.
5. The desktop PC quits the remote session by sending the “RemoteEnd" command.

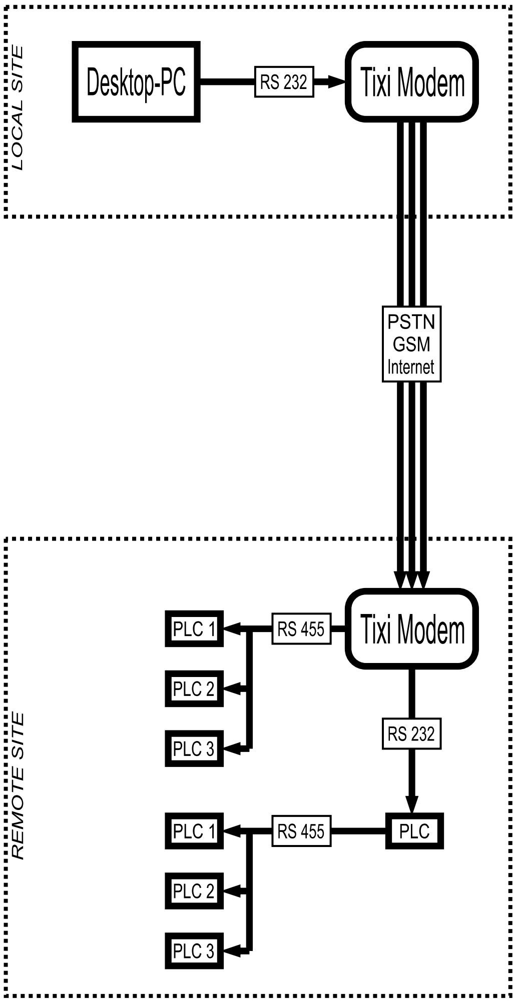
For an example, see the description of the `Remote` command in chapter 2.4.7.7.

5.3 Remote Control of an attached PLC

The following picture shows the configuration where the Embedded Control (Client Device) is controlled by the desktop PC. In this case the custom control protocol of the client device is used. The connection to the client device is established in two main steps.

1. Establish a remote controlling connection to the remote Tixi Alarm Modem as described above.
2. Send the **TransMode command** to switch the remote Tixi Alarm Modem to the Modem Mode.

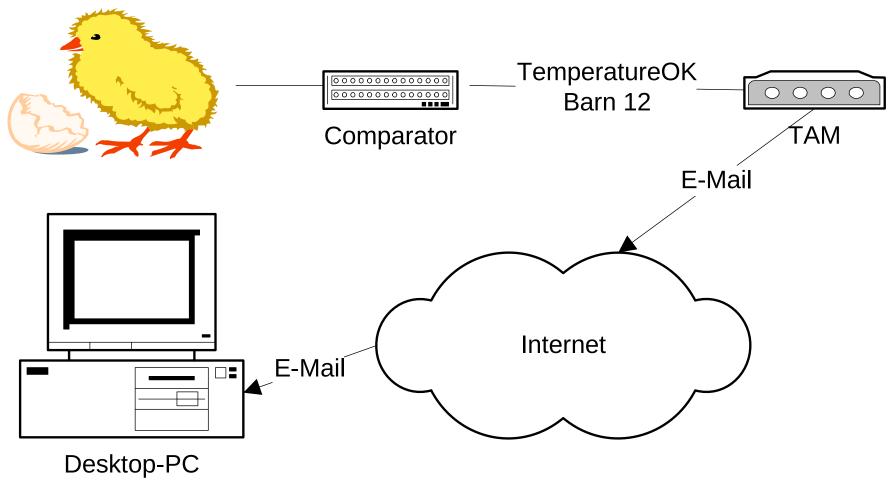
The client device can now be controlled by the desktop PC by means of the custom control protocol of the client device.
Such a connection can only be closed by a modem disconnection. To do this, the desktop PC sends the modem escape sequence ("+++") or the line is interrupted, e.g. by pulling out the cable.



6 Process I/O Ports and Variables

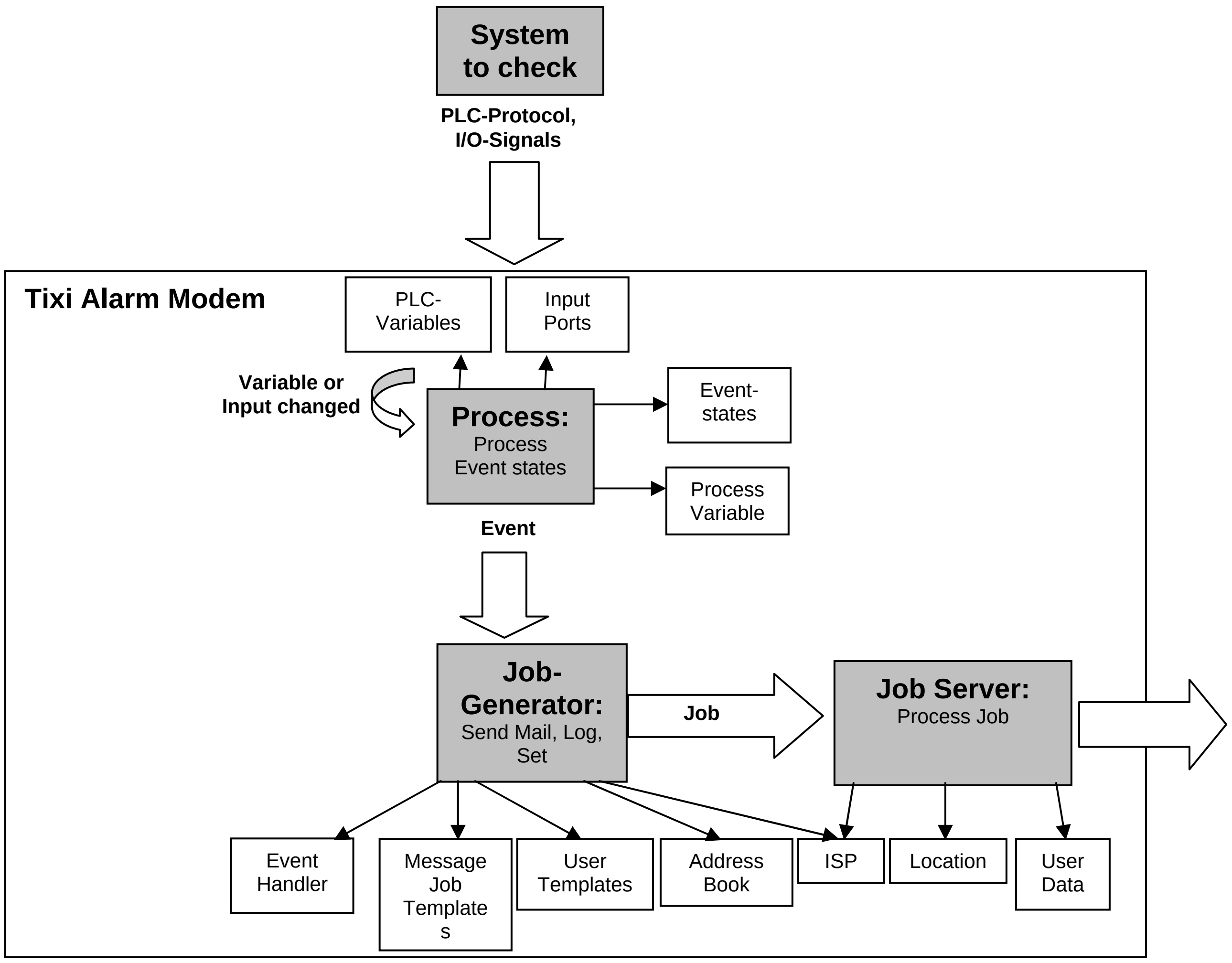
6.1 Introduction

As an option for the Tixi Alarm Modem some extension cards (Aluline) or modules (HutLine) can be added to the system. One possibility is the addition of an extension module with a certain number of digital inputs. These inputs can be used to signal states of connected systems.



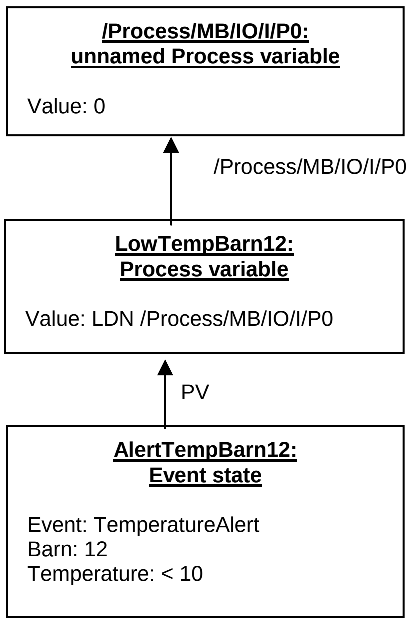
The 'Chicken Farm' example shows the new system configuration. The system to be checked prepares a digital signal which signals a system state. In the example this is the state 'temperature OK at barn 12'. The signal is set at an input port of the Tixi Alarm Modem. Tixi Alarm Modem now has the additional task of creating an event when the state of the system to be checked changes in a way that a message must be created. In the basic system (without extension module) configuration this part is done by the client.

To handle the changes of the input ports the 'Process' subsystem is part of the Tixi Alarm Modem firmware. The following picture shows how it replaces the role of the client in the basic configuration.



The process detects the changes of the input ports. This leads to a processing of the Event States which define the events to be created. The events are transmitted to the Job Generator which creates the jobs in the same way as if the events were received as messages from a client.

The following example shows the dependencies between the configuration items the process uses to generate an event:



The state of the input port is mapped to an unnamed process variable - in the example '/Process/MB/IO/I/P0' addressing the first bit of the input port of the HutLine mainboard.

Chapter 11 contains an overview of the correct IO addresses according to the different Tixi Alarm Modem hardware layouts.

The value is read from the port when it goes from 1 to 0 (this starts the processing of the input port change). There is a logical process variable defined with the name 'LowTempBarn12'. Its value is defined by the negation of the input port value (in this case the result is TRUE). It is possible to combine some input ports to a process variable by calculating its value from some different input port values (e.g. make an AND relation between two input ports). In this way meaningful variables can be created.

The EventState 'AlertTempBarn12' defines, that the Event 'TemperatureAlert' is created when the process variable 'LowTempBarn12' goes from FALSE to TRUE. If so, the event is created while the parameter of the event context is set to Barn: 12 and Temperature: <10. Now the event is processed in the same way as if sent by a client as message.

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

The EventStates are variables which remember whether an event was already activated or not. Therefore, only the change 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 before and then from FALSE to TRUE again.

Note: An open input port is indicated by a “1", a closed input port by a “0".

6.2 Define Process Variables

Typically an alert system like Tixi Alarm Modem 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 the process variable can be a manually set value or calculated from the values of the input ports. In this way, for example, a signal input can be linked with the input that indicates that the process power is on, so the input signal is only valid if both conditions are met.

A Process Variable value will only be calculated during EventStates processing, “Get” command or “LD” in another Process Variable.
Therefore you have to pay attention on the command order inside an instruction list.

We recommend to create not more than 30 calculated ProcessVars to keep system performance in an acceptable range.

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 in which it’s possible to write integer values or strings (up to 20 characters). Use of 20 independent variables is possible. Default value is option.

Reformating of the variable is possible (similar to PLC variable format).

If calculated 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 the configuration for the process variable 'LowTempBarn12' and the independent variable “Temperature" :

Database path: /PROCCFG/ProcessVars

```
<ProcessVars>
   <LowTempBarn12>                          Process Variable Group name
      <Value>
         <LDN _="/Process/MB/IO/I/P0"/>
      </Value>                              Calculate Value
   </LowTempBarn12>

   <Temperature/>
                                            Independent variable
</ProcessVars>
```

Process Variable Configuration

Syntax:

```
<ProcessVariableName exp="Exponent" format="FormatString">
   <Value>
      Instruction List
   </Value>
</ProcessVariableName>
```

Description:

Attribute group which defines a process variable.
The data type of calculated process variables is Int32 (see 6.7.1)

Elements:

ProcessVariableName:

Name of the process variable. It must be unique in the system.

Exponent:

Exponent of base 10 to specify fix point precision of
simpleType = Uint8, Uint16, Uint32, Int8, Int16, Int32 (see 6.7.1).
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

FormatString:

String that defines the value output format.
For a list of available format option see chapter 6.7.

Instruction List

List of instructions calculating the value of the process variable.

Example:

Process variable configuration that signals a low temperature in barn 12. The value is calculated by reading the negated value of bit 0 from the Alarm Modem HutLine mainboard.

```
<LowTempBarn12>
   <Value>
      <LDN _="/Process/MB/IO/I/P0"/>
   </Value>
</LowTempBarn12>
```

Process Variable loading a Tixi Alarm Modem analog value to reformat the displayed value.

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

Process Variable loading a Tixi Alarm Modem analog value with exponent.

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

Independent prozess variable with default value.

```
   <Variable1 def="250">
```

6.2.1 Instruction List

The Instruction List is a 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

```
   <ProcessVars>
      <LowTempBarn12>
         <Value>
            <LDN _="/Process/MB/IO/I/P0"/>
         </Value>
      </LowTempBarn12>
   </ProcessVars>
```

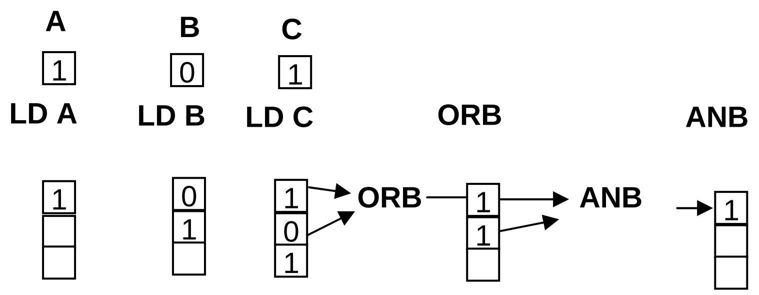
Instruction List

The instructions are adding or replacing the top item of the stack. The operations (except LD, LDN, NOT, TIME, MID, D_ON, D_OFF) are processing first two items of the stack, removing them and replacing the top item with its result value.

The following example shows the stack operation:

```
LD   A      ; Load Bit A
LD   B      ; Load Bit B
LD   C      ; Load Bit C
ORB         ; C or B
ANB         ; (C or B) and A
```

This program reads the three bit variables A, B and C. Then C and B are combined by an OR operation. The result is combined with A by an AND operation. The result is 1.



Currently, the stack has a size of 10 items. If an error occurs (stackoverflow, stackunderflow) the result value is 0.

The following Instruction list commands are implemented:

6.2.1.1 logical instructions

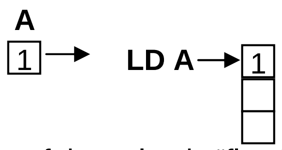
***LD Instruction** - Load value*

Syntax:

```
    <LD _="BitAddress"/>
    <LD _="Address"/>
```

Description:

Instruction. It reads the value defined by the address or addresses and loads it at the top of the processing stack.



If the type of the value is “float”, only the fixed part is loaded.

Elements:

BitAddress:

see chapter 6.2.1.9.

Address:

see chapter 6.2.1.9.

Examples:

Load the bit 4 of the bit field of the module with the address 40.

```
    <LD _="/Process/C40/I/P4"/>
```

Load the value of register R100.

```
    <LD _="/Process/Bus1/Device_0/R100"/>
```

Load the registers R100 to R102.

```
    <LD v1="/Process/Bus1/Device_0/R100"
    v2="/Process/Bus1/Device_0/R101"
    v3="/Process/Bus1/Device_0/R102"/>
```

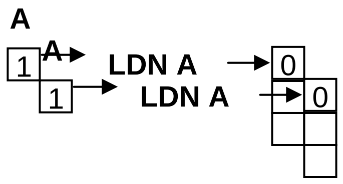
***LDN/DLDN Instruction** - Load Bit and Negate / Load Bit and Negate (binary)*

Syntax:

```
    <LDN _="BitAddress"/>
    <DLDN _="Address"/>  (@FW 2.2)
```

Description:

Instruction. It reads the value defined by the address and loads its negated value at the top of the processing stack.



Elements:

BitAddress:

see chapter 6.2.1.9.

Address:

see chapter 6.2.1.9.

Examples:
Load the negated bit 4 of the bit field of the module with the address 40.

```
    <LDN _="/Process/C40/I/P4"/>
```

Load the binary negated value of 2 (binary result = ”-3”):

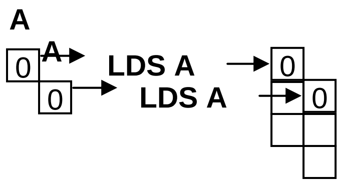
```
    <DLDN _="2"/>
```

***LDS Instruction** - Load Special* (@FW 2.2)

Syntax:

```
    <LDS _="PLCVariableAddress" AddInfo="ErrorCode"/>
```

Description:
Instruction. It reads the additional info of a PLC variable defined by the address and loads its 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:
PLCVariableAddress:
see PLC-TiXML- Manual
ErrorCode:
see PLC-TiXML- Manual
Examples:
Load the ErrorClass and ErrorNumber information of the PLC variable “Variable_0” of “Device_0” on PLC-Bus “Bus1” on the stack.

```
    <LDS _="/Process/Bus1/Device_0/Variable_0" AddInfo=”Error”/>
```

***AND/DAND Instruction** - Load + AND / Load + AND (binary)*

Syntax:

```
    <AND v1="BitAddress" v2="BitAddress"/>
    <DAND v1="Address" v2="Address"/> (@FW 2.2)
```

Description:
Instruction. Combination of the instructions LD and AND (DAND). First, it reads the value defined by the address and loads its value at the top of the processing stack. Second, process a binary AND operation between the two first stack items and replace both by the result of the operation.
Elements:
BitAddress:
see chapter 6.2.1.9.
Address:
see chapter 6.2.1.9.
Examples:
Load the bits 4 and 2 of the bit field of the module with the address 40, and set the AND operation result.

```
    <AND v1=”/Process/C40/I/P2” v2=”/Process/C40/I/P4”/>
```

or

```
   <LD _="/Process/C40/I/P2"/>
   <AND _="/Process/C40/I/P4"/>
```
or
```
   <LD _="/Process/C40/I/P2"/>
   <LD _="/Process/C40/I/P4"/>
   <AND/>
```
Load the values 2 and 3 and sets the binary AND operation result (binary result = “2”).
```
   <DAND v1=”2” v2=”3”/>
```

ANDN/DANDN Instruction *- Load Negate+ AND / Load Negate+ AND (binary)*

Syntax:
```
   <ANDN v1="BitAddress" v2="BitAddress"/>
   <DANDN v1="Address" v2="Address"/> (@FW 2.2)
```
Description:
Instruction. Combination of the instructions LDN (DLDN) and AND (DAND). First, it reads the negated value defined by the address and loads its value at the top of the processing stack. Second, process a binary AND operation between the two first stack items and replace both by the result of the operation.

Elements:
BitAddress:
see chapter 6.2.1.9.
Address:
see chapter 6.2.1.9.

Examples:
Load the negated bit 4 and the bit 2 bit of the bit field of the module with the address 40, and set the AND operation result.
```
   <LD _="/Process/C40/I/P2"/>
   <ANDN _="/Process/C40/I/P4"/>
```
Load the bit 4 and bit 2, negate both and set the AND operation result.
```
   <ANDN v1=”/Process/C40/I/P2” v2=”/Process/C40/I/P4”/>
```
Load the bit 4 and bit 2, negate both and set the AND operation result.
```
   <ANDN v1=”/Process/C40/I/P2” v2=”/Process/C40/I/P4”/>
```
Load the values 3 and 2, negate both and set the AND operation result (binary result = “-4”).
```
   <DANDN v1=”3” v2=”2”/>
```

OR/DOR Instruction *- Load + OR / Load + OR (binary)*

Syntax:
```
   <OR v1="BitAddress" v2="BitAddress"/>
   <DOR v1="Address" v2="Address"/> (@FW 2.2)
```
Description:
Instruction. Combination of the instructions LD and OR (DOR). First, it reads the value defined by the address and loads its value at the top of the processing stack. Second, process a binary OR operation between the two first stack items and replace both by the result of the operation.

Elements:
BitAddress:
see chapter 6.2.1.9.
Address:
see chapter 6.2.1.9.

Examples:

Load the bits 4 and 2 of the bit field of the module with the address 40, and set the OR operation result.

```
   <OR v1=”/Process/C40/I/P2” v2=”/Process/C40/I/P4”/>
```

or

```
   <LD _="/Process/C40/I/P2"/>
   <OR _="/Process/C40/I/P4"/>
```

or

```
     <LD _="/Process/C40/I/P2"/>
     <LD _="/Process/C40/I/P4"/>
     <OR/>
```

Load the values 3 and 2 and set the OR operation result (binary result = “3”).

```
   <DOR v1=”3” v2=”2”/>
```

ORN/DORN Instruction *- Load Negate + OR / Load Negate + OR (binary)*

Syntax:

```
     <ORN _="BitAddress"/>
     <DORN _="Address"/>  (@FW 2.2)
```

Description:

Instruction. Combination of the instructions LDN and OR (DOR). First, it reads the negated value defined by the address and loads its value at the top of the processing stack. Second, process a binary OR operation between the two first stack items and replace both by the result of the operation.

Elements:

BitAddress:

see chapter 6.2.1.9.

Address:

see chapter 6.2.1.9.

Examples:

Load the negated bit 4 and the bit 2 of the bit field of the module with the address 40, and set the OR operation result.

```
   <LD _="/Process/C40/I/P2"/>
   <ORN _="/Process/C40/I/P4"/>
```

Load the bit 4 and bit 2, negate both and set the OR operation result.

```
   <ORN v1=”/Process/C40/I/P2” v2=”/Process/C40/I/P4”/>
```

Load the values 3 and 2, negate both and set the OR operation result (binary result = “-3”).

```
   <DORN v1=”3” v2=”2”/>
```

XOR/DXOR Instruction *- Load + XOR / Load + XOR (binary)*

Syntax:

```
   <XOR v1="BitAddress" v2="BitAddress"/>
   <XOR v1="Address" v2="Address"/>  (@FW 2.2)
```

Description:

Instruction. Combination of the instructions LD and XOR (DXOR). First, it reads the value defined by the address and loads its value at the top of the processing stack. Second, process a binary XOR operation between the two first stack items and replace both by the result of the operation.

Elements:

BitAddress:

see chapter 6.2.1.9.

Address:

see chapter 6.2.1.9.

Examples:
Load the bits 4 and 2 of the bit field of the module with the address 40, and set the XOR operation result.

```
    <XOR v1=”/Process/C40/I/P2” v2=”/Process/C40/I/P4”/>
```

or

```
    <LD _="/Process/C40/I/P2"/>
    <XOR _="/Process/C40/I/P4"/>
```

or

```
    <LD _="/Process/C40/I/P2"/>
    <LD _="/Process/C40/I/P4"/>
    <XOR/>
```

Load the values 3 and 2 and set the XOR operation result (binary result =”1”).

```
    <DXOR v1=”3” v2=”2”/>
```

XORN/DXORN Instruction *- Load Negate + XOR / Load Negate + XOR (binary)*

Syntax:

```
    <XORN _="BitAddress"/>
    <DXORN _="Address"/> (@FW 2.2)
```

Description:
Instruction. Combination of the instructions LDN (DLDN) and XOR (DXOR). First, it reads the negated value defined by the address and loads its value at the top of the processing stack. Second, process a binary XOR operation between the two first stack items and replace both by the result of the operation.

Elements:
BitAddress:
see chapter 6.2.1.9.
Address:
see chapter 6.2.1.9.

Examples:
Load the negated bit 4 and the bit 2 of the bit field of the module with the address 40, and set the OR operation result.

```
    <LD _="/Process/C40/I/P2"/>
    <XORN _="/Process/C40/I/P4"/>
```

Load the bit 4 and bit 2, negate both and set the XOR operation result.

```
    <XORN v1=”/Process/C40/I/P2” v2=”/Process/C40/I/P4”/>
```

Load the bit 3 and bit 2, negate both and set the XOR operation result (binary result = “1”).

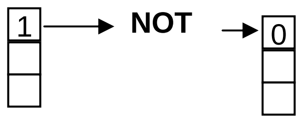
```
    <DXORN v1=”3” v2=”2”/>
```

NOT Instruction *– Negation*

Syntax:

```
    <NOT/>
```

Description:
Instruction. Process a NOT operation of the first stack item.



Elements:
empty

Examples:
Load the bit 2 of the bit field of the module with the address 40, and negate it.

```
<LD _="/Process/C40/I/P2"/>
<NOT/>
```

same as

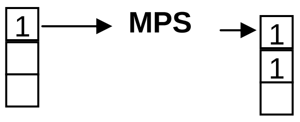
```
<LDN _="/Process/C40/I/P2"/>
```

***MPS Instruction** – Multiply stack* (@FW 2.0)

Syntax:

```
<MPS/>
```

Description:
Instruction. Multiplies a stack value to use it for two operations.



Elements:
empty

Examples:
Load the bit 2 of the bit field of the module with the address 40, multiply it and set two output ports.

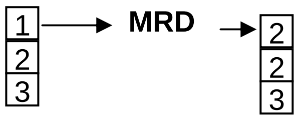
```
<LD _="/Process/C40/I/P2"/>
<MPS/>
<ST _="/Process/C40/Q/P0"/>
<ST _="/Process/C40/Q/P1"/>
```

***MRD Instruction** – Copy 2nd stack level to top of stack* (@FW 2.2)

Syntax:

```
<MRD/>
```

Description:
Instruction. Replaces the value on the top of the stack with the value from the second stack level.



Elements:
empty

Examples:
Load 1 and 2 on the stack. The result is 2 because MRD replaces the 2 at the top of the stack by the 1 from the second level.

```
<LD _="1"/>
<LD _="2"/>
<MRD/>
<ADD/>
```

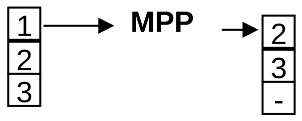
MPP Instruction *– Remove the value at the top of stack* (@FW 2.2)

Syntax:

```
    <MPP/>
```

Description:

Instruction. Removes the value on the top of the stack.



Elements:

empty

Examples:

The value from Port 1 of the module with address 40 is removed from the stack. The value from Port 0 of the module with address 40 is stored into the output port 0 of module 40.

```
      <LD _="/Process/C40/I/P0"/>
      <LD _="/Process/C40/I/P1"/>
      <MPP/>
      <ST _="/Process/C40/Q/P0"/>
```

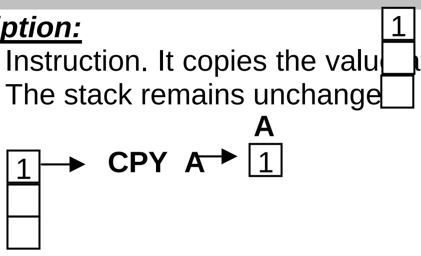
CPY Instruction – *Copy value* (@FW 2.2)

Syntax:

```
      <CPY _="BitAddress"/>
      <CPY _="Address"/>
```

Description:

Instruction. It copies the value at the top of the processing stack to the given address. The stack remains unchanged.



Elements:

BitAddress:

see chapter 6.2.1.9.

Address:

see chapter 6.2.1.9.

Examples:

Copy the value of the input P4 into the output P4.

```
      <LD _="/Process/C40/I/P4"/>
      <CPY _="/Process/C40/Q/P4"/>
```

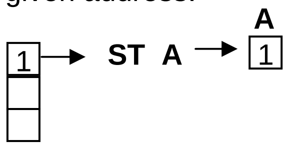
ST Instruction - Store

Syntax:

```
      <ST _="BitAddress"/>
      <ST _="Address"/>
```

Description:

Instruction. It stores (moves) the value from the top of the processing stack into the given address.



Elements:

BitAddress:

see chapter 6.2.1.9.

Address:

see chapter 6.2.1.9.

Examples:

Store the value of the input P4 into the output P4.

```
<LD _="/Process/C40/I/P4"/>
<ST _="/Process/C40/Q/P4"/>
```

6.2.1.2 Comparison instructions

GT/GTI Instruction – *greater than / greater than (integer)* (@FW 2.0)

Syntax:

```
<GT v1="value1" v2="value2"/>
<GTI v1="value1" v2="value2"/>
```

Description:

Instruction. Compares both values and if value1 is greater than value2 it stores a 1 at the top of the processing stack.

Elements:

value1:

address of the first value. see chapter 6.2.1.9.

value2:

address of the value to compare with, or value. see chapter 6.2.1.9.

Examples:

Compares register R100 with value 100.

```
<GT v1="/Process/Bus1/Device_0/R100" v2="100"/>
```

or

```
<LD _="/Process/Bus1/Device_0/R100"/>
<GT _="100"/>
```

or

```
<LD _="/Process/Bus1/Device_0/R100"/>
<LD _="100"/>
<GT/>
```

Compares integer register R200 with value -100.

```
<GTI v1="/Process/Bus1/Device_0/R200" v2="-100"/>
```

Compares register R100 with register R150 :

```
<GT v1="/Process/Bus1/Device_0/R100"
v2="/Process/Bus1/Device_0/R150"/>
```

LT/LTI Instruction – less than / less than (integer) (@FW 2.0)

Syntax:

```
<LT v1="value1" v2="value2"/>
<LTI v1="value1" v2="value2"/>
```

Description:

Instruction. Compares both values and if value1 is less than value2 it stores a 1 at the top of the processing stack.

Elements:

value1:

address of the first value. see chapter 6.2.1.9.

value2:

address of the value to compare with, or value. see chapter 6.2.1.9.

Examples:

Compares register R100 with value 100.

```
<LT v1="/Process/Bus1/Device_0/R100" v2="100"/>
```

or

```
<LD _="/Process/Bus1/Device_0/R100"/>
<LT _="100"/>
```

or

```
<LD _="/Process/Bus1/Device_0/R100"/>
<LD _="100"/>
<LT/>
```

Compares integer register R200 with value -100.

```
<LTI v1="/Process/Bus1/Device_0/R200" v2="-100"/>
```

Compares register R100 with register R150 :

```
 <LT v1="/Process/Bus1/Device_0/R100"
v2="/Process/Bus1/Device_0/R150"/>
```

EQ Instruction – equal (@FW 2.0)

Syntax:

```
<EQ v1="value1" v2="value2"/>
```

Description:

Instruction. Compares both values and if value1 is equal value2 it stores a 1 at the top of the processing stack.

Elements:

value1:

address of the first value. see chapter 6.2.1.9.

value2:

address of the value to compare with, or value. see chapter 6.2.1.9.

Examples:

Compares register R100 with value 100.

```
<EQ v1="/Process/Bus1/Device_0/R100" v2="100"/>
```

or

```
<LD _="/Process/Bus1/Device_0/R100"/>
<EQ _="100"/>
```

or

```
<LD _="/Process/Bus1/Device_0/R100"/>
<LD _="100"/>
<EQ/>
```

Compares register R100 with register R150 :

```
<EQ v1="/Process/Bus1/Device_0/R100"
v2="/Process/Bus1/Device_0/R150"/>
```

NE Instruction – not equal (@FW 2.0)

Syntax:

```
<NE v1="value1" v2="value2"/>
```

Description:

Instruction. Compares both values and if value1 is not equal value2 it stores a 1 at the top of the processing stack.

Elements:

value1:

address of the first value. see chapter 6.2.1.9.

value2:

address of the value to compare with, or value. see chapter 6.2.1.9.

Examples:

Compares register R100 with value 100.

```
<NE v1="/Process/Bus1/Device_0/R100" v2="100"/>
```

or

```
<LD _="/Process/Bus1/Device_0/R100"/>
<NE _="100"/>
```

or

```
<LD _="/Process/Bus1/Device_0/R100"/>
<LD _="100"/>
<NE/>
```

Compares register R100 with register R150 :

```
<NE v1="/Process/Bus1/Device_0/R100"
v2="/Process/Bus1/Device_0/R150"/>
```

GE/GEI Instruction – greater equal / greater equal (integer) (@FW 2.0)

Syntax:

```
<GE v1="value1" v2="value2"/>
<GEI v1="value1" v2="value2"/>
```

Description:

Instruction. Compares both values and if value1 is equal or greater than value2 it stores a 1 at the top of the processing stack.

Elements:

value1:

address of the first value. see chapter 6.2.1.9.

value2:

address of the value to compare with, or value. see chapter 6.2.1.9.

Examples:
Compares register R100 with value 100.

```
<GE v1="/Process/Bus1/Device_0/R100" v2="100"/>
```

or

```
<LD _="/Process/Bus1/Device_0/R100"/>
<GE _="100"/>
```

or

```
<LD _="/Process/Bus1/Device_0/R100"/>
<LD _="100"/>
<GE/>
```

Compares integer register R200 with value -100.

```
<GEI v1="/Process/Bus1/Device_0/R200" v2="-100"/>
```

Compares register R100 with register R150 :

```
<GE v1="/Process/Bus1/Device_0/R100"
v2="/Process/Bus1/Device_0/R150"/>
```

LE/LEI Instruction – *less equal / less equal (integer)* (@FW 2.0)

Syntax:

```
<LE v1="value1" v2="value2"/>
<LEI v1="value1" v2="value2"/>
```

Description:

Instruction. Compares both values and if value1 is equal or less than value2 it stores a 1 at the top of the processing stack.

Elements:

value1:

address of the first value. see chapter 6.2.1.9.

value2:

address of the value to compare with, or value. see chapter 6.2.1.9.

Examples:
Compares register R100 with value 100.

```
<LE v1="/Process/Bus1/Device_0/R100" v2="100"/>
```

or

```
<LD _="/Process/Bus1/Device_0/R100"/>
<LE _="100"/>
```

or

```
<LD _="/Process/Bus1/Device_0/R100"/>
<LD _="100"/>
<LE/>
```

Compares register R100 with register R150 :

```
<LE v1="/Process/Bus1/Device_0/R100"
v2="/Process/Bus1/Device_0/R150"/>
```

6.2.1.3 Bit mask instruction

***MSK/DMSK Instruction –** bit mask* (@FW 2.0)

Syntax:

```
<MSK v1="value" v2="mask"/>
<DMSK v1="value" v2="mask"/>
```

Description:

Instruction.
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.
DMSK is used to give the sum of all bits set within the mask.

Elements:

value:

address of the value to mask. see chapter 6.2.1.9.

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…

Examples

Masks register R100 with mask 7. If al least one of the first three bits is set, the Prozessvariable becomes “1”.

```
<MSK v1="/Process/Bus1/Device_0/R100" v2="7"/>
```

Mask register R100 with mask 3. The result of the ProcessVariable will be the sum of all set bits covered by the mask.

```
<DMSK v1="/Process/Bus1/Device_0/R100" v2="3"/>
```

6.2.1.4 math operations

***ADD/ADDI Instruction –** addition / addition (integer)* (@FW 2.2)

Syntax:

```
<ADD v1="value1" v2="value2"/>
<ADDI v1="value1" v2="value2"/>
```

Description:

Instruction. Adds value2 to value1.

Elements:

value1:

address of the first value. see chapter 6.2.1.9.

value2:

address of the value, or value to add to value1. see chapter 6.2.1.9.

Examples:

Load register R100 and R101 and add R101 to R100

```
<ADD v1=”/Process/Bus1/Device_0/R100”
v2=”/Process/Bus1/Device_0/R101”/>
```

or

```
<LD _=" /Process/Bus1/Device_0/R100"/>
<ADD _=”/Process/Bus1/Device_0/R101”/>
```

or

```
<LD _="/Process/Bus1/Device_0/R100"/>
<LD _="/Process/Bus1/Device_0/R101"/>
<ADD/>
```

SUB/SUBI Instruction – *subtraction / subtraction (integer)* (@FW 2.2)

Syntax:

```
<SUB v1="value1" v2="value2"/>
<SUBI v1="value1" v2="value2"/>
```

Description:

Instruction. Subtracts value2 from value1.

Elements:

value1:

address of the first value. see chapter 6.2.1.9.

value2:

address of the value, or value to subtract from value1. see chapter 6.2.1.9.

Examples:

Load register R100 and R101 and subtract R101 from R100

```
<SUB v1=”/Process/Bus1/Device_0/R100”
v2=”/Process/Bus1/Device_0/R101”/>
```

or

```
<LD _="/Process/Bus1/Device_0/R100"/>
<SUB _=”/Process/Bus1/Device_0/R101”/>
```

or

```
<LD _="/Process/Bus1/Device_0/R100"/>
<LD _="/Process/Bus1/Device_0/R101"/>
<SUB/>
```

MUL/MULI Instruction – *multiplication / multiplication (integer)* (@FW 2.2)

Syntax:

```
<MUL v1="value1" v2="value2"/>
<MULI v1="value1" v2="value2"/>
```

Description:

Instruction. Multiplicates value1 with value2.

Elements:

value1:

address of the first value multiplicated by value2. see chapter 6.2.1.9.

value2:

address of the multiplicator, or multiplicator. see chapter 6.2.1.9.

Examples:

Load register R100 and R101 and multiplicates R100 with R101

```
<MUL v1=”/Process/Bus1/Device_0/R100”
v2=”/Process/Bus1/Device_0/R101”/>
```

or

```
<LD _="/Process/Bus1/Device_0/R100"/>
<MUL _=”/Process/Bus1/Device_0/R101”/>
```

or

```
<LD _="/Process/Bus1/Device_0/R100"/>
<LD _="/Process/Bus1/Device_0/R101"/>
<MUL/>
```

DIV/DIVI Instruction – division / division (integer) (@FW 2.2)

Syntax:

```
<DIV v1="value1" v2="value2"/>
<DIVI v1="value1" v2="value2"/>
```

Description:

Instruction. Divides value1 by value2.

Elements:

value1:

address of the first value divided by value2. see chapter 6.2.1.9.

value2:

address of the divisor, or divisor. see chapter 6.2.1.9.

Examples:

Load register R100 and R101 and divide R100 by R101

```
<DIV v1="/Process/Bus1/Device_0/R100"
v2="/Process/Bus1/Device_0/R101"/>
```

or

```
<LD _="/Process/Bus1/Device_0/R100"/>
<DIV _="/Process/Bus1/Device_0/R101"/>
```

or

```
<LD _="/Process/Bus1/Device_0/R100"/>
<LD _="/Process/Bus1/Device_0/R101"/>
<DIV/>
```

6.2.1.5 TIME instruction

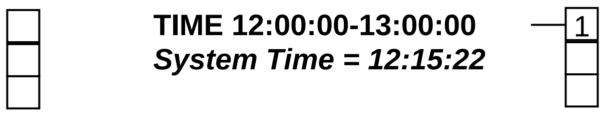
***TIME Instruction** - Compare Time Span with current Time*

Syntax:

```
<TIME _="YYYY/MM/DD,HH:MM:SS-YYYY/MM/DD,HH:MM:SS"/>
```

Description:

Instruction. Check whether the current system time of day is inside the given time span or not. Add the value '1' at the first stack item when the system time lies within the time span. Add the value '0' at the first stack item when the system time lies outside the time span. The same value is inserted when an invalid time value is typed in.



Elements:

YYYY:

the year

MM:

the month

DD:

the day

HH:

***00..23** hours in the day.*

MM:

***00..59** minutes in the hour.*

SS:

***00..59** seconds in the minute.*

Examples:
Check the time span between 07:00 and 12:00.
```
      <TIME _="07:00:00-12:00:00"/>
```

Check the time span between 23:00 and 07:00 at the next day.
```
      <TIME _="23:00:00-07:00:00"/>
```

Check the time 05:00.
```
      <TIME _="05:00:00-05:02:00"/>
```

6.2.1.6 Power-on/off delay instruction

D_ON / D_OFF Instruction *– Power-on/off delay*

Syntax:
```
      <D_ON time="X"/>
      <D_OFF time="X"/>
```

Description:
Instruction. Waits the defined time to accept a status as true.

Elements:
X: Time to wait
possible values:
“Xs" for seconds, e.g. 10s
“Xm" for minutes, e.g. 5m
“Xh" for hours, e.g. 1h
“Xd" for days, e.g. 7d

Examples:
If input port P0 is set at least 10s, output P0 will be set.
```
    <LD _="/Process/MB/IO/I/P0"/>
    <D_ON time="10s"/>
    <MPS/>
    <ST _="/Process/MB/IO/Q/P0"/>
```

6.2.1.7 FIND_BIT_ADDRESS instruction

In most cases an alarm is triggered by a bit variable. Because of the Alarm Modem EventStates limited to 100 entries, only 100 different alarms would be possible with common alarm projects. On larger PLC systems much more alarms are required. These systems do offer the alarm bits within word or dword variables.

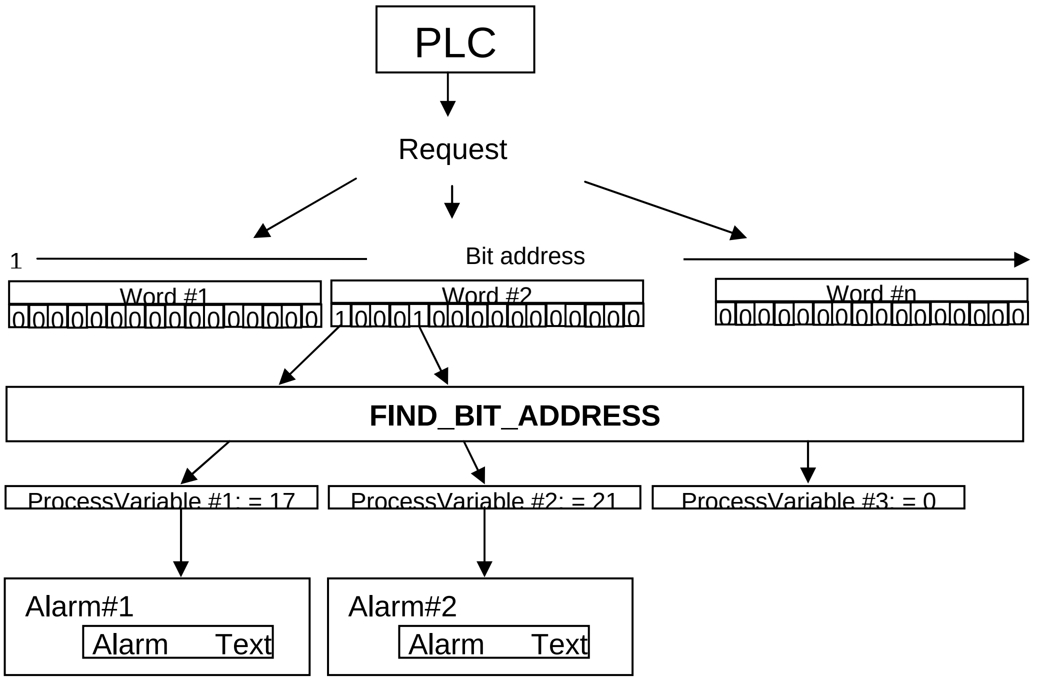
With the MSK instruction (chapter 6.2.1.3) it is possible to trigger an alarm if any of the bits is set, but it isn’t possible to select a message text depending on the bit.

Therefore a instruction is required that finds alarm bits within word or dword variables and is also able to tell the job processor which bit was set.

Requirements:

- Up to 7 dwords are composed in a certain order and the modem counts the bits starting with the lowest bit of the word 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 simultaneous set bits only **the first three** will be detected. The bits are counted **started by one**.

- The alarm messages template is the same for all alarm bits, but the message text is exchanged via reference depending on the bit address.



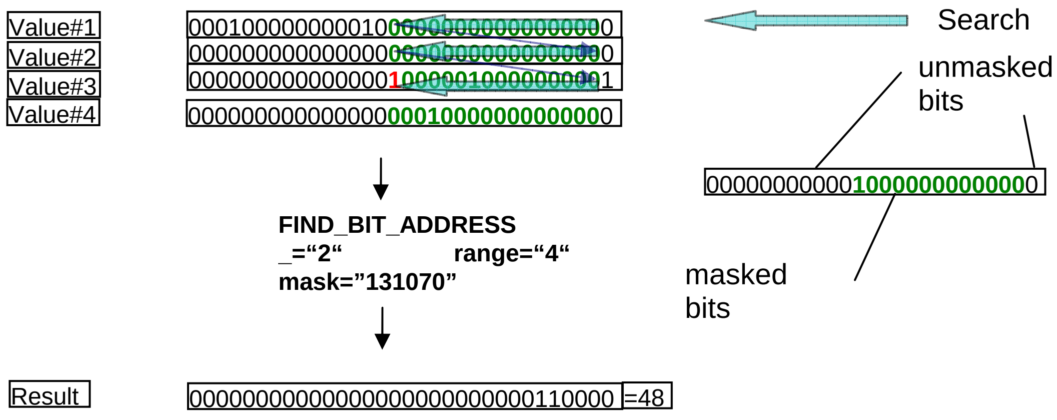
***FIND_BIT_ADDRESS Instruction** – Find the address of the n'th bit set* (@FW 2.2)

Syntax:

```
<FIND_BIT_ADDRESS _="BitRank"
range="NumberOfStackEntriesToScan" mask = "CountMask"/>
```

Description:

Instruction. 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,*** beginning with 1. If such a bit is found, write the counter result at the top of the stack, otherwise write a 0 at the top of the stack. Remove all ***NumberOfStackEntriesToScan* +** 3 Entries from the stack.



Elements:

BitRank:

First, second, third…Bit to find (1… 4294967295).

NumberOfStackEntriesToScan:

Number of values on the stack, which are to be scanned (1….7)

Attention:

Before you call the" Find_Bit_Address" instruction, load the number of variables specified in the "range" parameter on the stack (see example above), otherwise the instruction results in an error.

CountMask:

Masks the bits to count in a stack value.
Maskbit =1 count the bit.
Maskbit =0 do not count the bit.

Example:
To count only the bits of byte values use a mask of
255 = 00000000000000000000000011111111.

***Examples**:*

Define two process variables.
Load four status dwords from a PLC and find the first bit set in the first process variable and the second bit set in the second variable.

Mask with 131070 = 00000000000000011111111111111110

```
<ProcessVars>
   <P305_308_01>
      <Value>
         <LD _="/Process/Bus1/D1/P308"/>
         <LD _="/Process/Bus1/D1/P307"/>
         <LD _="/Process/Bus1/D1/P306"/>
         <LD _="/Process/Bus1/D1/P305"/>
         <FIND_BIT_ADDRESS _="1" range="4" mask="131070" />
   </Value>
   </P305_308_01>
   <P305_308_02>
      <Value>
         <LD _="/Process/Bus1/D1/P308"/>
         <LD _="/Process/Bus1/D1/P307"/>
         <LD _="/Process/Bus1/D1/P306"/>
         <LD _="/Process/Bus1/D1/P305"/>
         <FIND_BIT_ADDRESS _="2" range="4" mask="131070" />
   </Value>
   </P305_308_02>
</ProcessVars>
```

6.2.1.8 Text parser instruction

***MID Instruction** – text parser* (@FW 2.2)

Syntax:

```
<MID _="string" start="X" length="Y"/>
```

Description:

Instruction. Loads part of a string from position start to length.

Note: The result has to persist of digits only.

Elements:

String:

String to parse. If string a BitAdress/Address (*see chapter 6.2.1.9.*), reference must be made using 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

Examples:

Extract the “hour” out of the Tixi Alarm Modem “Time” string:

```
<MID _="&#xae;/TIMES/Time" start="0" length="2"/>
```

or

```
<LD start="0"/>
<LD length="2"/>
<MID _="&#xae;/TIMES/Time"/>
```

6.2.1.9 Bit address / address

BitAddress

Syntax:

/Process/CCardAddress/[PortGroup]/PortType/PPortIndex

Description:

BitAddress. Addresses the input or output port of an extension card/module of the Tixi Alarm Modem. Port bits can be addressed in different ways. It is possible to address a single bit or sequences of bits of a port. 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

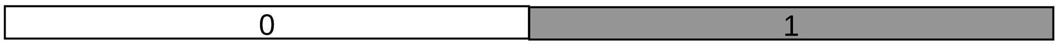
Single Bit Access:example: /Process/C0/I/P4

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

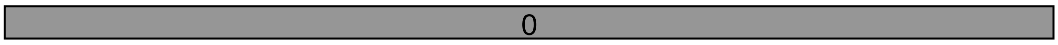
Byte Access: example: /Process/C0/IB/P2

0	1	2	3

Word Access: example: /Process/C0/IW/P1



Double Word Access: example /Process/C0/ID/P0



Elements:
The port is addressed by a slash divided path notation:

Process:	Process sub system path

CardAddress:

40...4E	HEX number corresponds to the setting of the address jumper of the HutLine extension modules.
0...15	Number corresponds to the setting of the address jumper of the Aluline extension card.

PortGroup: *(only on HutLine mainboard I/Os)*

IO	Group of digital in- and outputs
A	Group of analog inputs and counters

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.

Examples:
5. bit of the input bit field of the module with the address 40.

```
/Process/C40/I/P4
```

Address

Syntax:

/Process/BusID/DeviceID/VariableID
/Process/PV/ProcessVariable
/SystemProperty

Description:

Address. Addresses a process or system variable of the Tixi Alarm Modem or a variable of an attached PLC.
See PLC-TiXML-Manual for further information.

Elements:
The variable is addressed by a slash divided path notation:

Process: Process sub system path

BusID:
Name or ID of PLC bus, e.g. “`Bus1`”

DeviceID:
Name or ID of PLC station, e.g. “`Device_0`“,“`D0`“ or “`MyPLC`“ etc.

VariableID:
Name or ID of PLC variable, e.g. “Variable_0“ or “Marker1“ etc.

PV: Process Variable group path

ProcessVariable:
Name of process variable, e.g. “Alarm_0_PV“ etc.
SystemProperty:
Path to a Alarm Modem system variable. See appendix 12

<u>Examples</u>:
PLC Variable “Variable_0” on station “Device_0” at PLC bus “Bus1”:
```
      /Process/Bus1/Device_0/Variable_0
```

Process variable “Alarm_0_PV”:
```
       /Process/PV/Alarm_0_PV
```

System variable “FirstCycle”:
```
       /Process/MB/FirstCycle
```

System variable “GSM Account”:
```
       /GSM/Account
```

6.3 Define Event States

The event states define the events, which are sent to the job generator when the value of a process variable goes from '0' = FALSE to '1' = TRUE. Therefore, each event state is related to a process variable (but one process variable can be referred by more than one event state). Additionally, the event state can be disabled. So no event is generated when the process variable is changed.

The maximum amount of EventStates is 100.

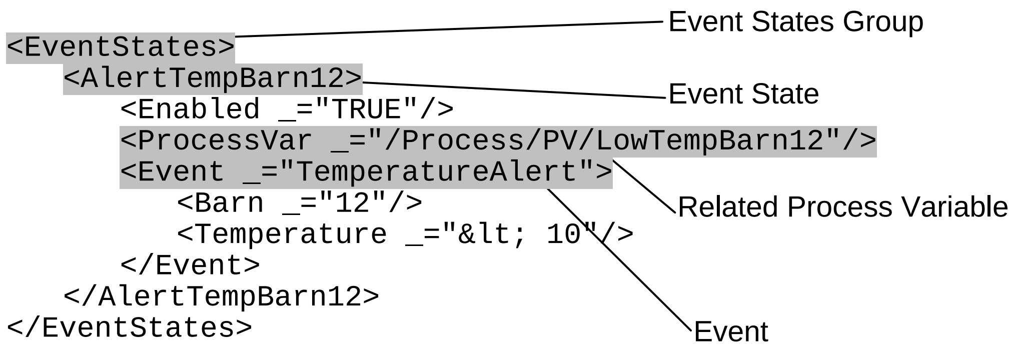
There are two different ways to assign an event to a process variable:

- With use of ProcessVars database (chapter 6.2) which is more flexible and allows logical operations and handling of bus variables.
- Without use of ProcessVars database. The advantage is much faster processing of event states.

6.3.1 Define Event States with use of ProcessVars database

The following example shows the configuration for the Event State 'LowTempBarn12' :

Database path: /PROCCFG/EventStates



```
<EventStates>
   <AlertTempBarn12>
      <Enabled _="TRUE"/>
      <ProcessVar _="/Process/PV/LowTempBarn12"/>
      <Event _="TemperatureAlert">
         <Barn _="12"/>
         <Temperature _="&lt; 10"/>
      </Event>
   </AlertTempBarn12>
</EventStates>
```

Event State Configuration

Syntax::

```
<EventStateName>
   <Enabled _="EnableState"/>
   <ProcessVar _="/Process/PV/relatedProcessVariable"
   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:

State valid after SetConfig or system start, no dynamic change possible.

TRUE	The event state is enabled, events are created.
FALSE	The event state is disabled, no events are created.
FAST	The Event is processed immediately after checking it’s EventState. (with TRUE a list of changed EventStates will be created first)
1	same as TRUE (@FW 2.0)
0	same as FALSE

relatedProcessVariable:

Name of the related process variable defined in the /PROCESS/PV group.

State:

high	Event is generated if assigned ProcessVariable result changes from 0 to 1
low	Event is generated if assigned ProcessVariable result changes from 1 to 0
both	Event is generated on every change of assigned ProcessVariable result

EventName:

Name of the event, generated when the process variable goes from 0 to 1.
Note: There must be an Event Handler defined of the same name.

ParameterList:

List of XML encoded parameter representing the event context. A parameter is written in a single XML- tag with:

```
<ParameterKey _="Value"/>
```

where

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

Example:

Event state configuration with the name 'LowTempBarn12' that creates a 'TemperatureAlert' event, when the process variable LowTempBarn12 goes from 0 to 1. The context parameter of the event is defined as Barn: 12 and Temperature < 10.

```
<LowTempBarn12>
      <Enabled _="TRUE"/>
      <ProcessVar _="/Process/PV/LowTempBarn12"/>
      <Event _="TemperatureAlert">
         <Barn _="12"/>
         <Temperature _="&lt;10"/>
      </Event>
</LowTempBarn12>
```

6.3.2 Define Event States without use of ProcessVars database

The following example shows the configuration for the Event State 'LowTempBarn12' :

Database path: /PROCCFG/EventStates

```
                                                   Event States Group
<EventStates>
   <AlertTempBarn12>                               Event State
      <Enabled _="TRUE"/>
      <ProcessVar _="/Process/MB/IO/I/P3" flank="high"/>
      <Event _="TemperatureAlert">
         <Barn _="12"/>                             Related Process Variable
         <Temperature _="&lt; 10"/>
      </Event>
   </AlertTempBarn12>
</EventStates>                                      Event
```

Event State Configuration

Syntax:

```
   <EventStateName>
      <Enabled _="EnableState"/>
      <ProcessVar _="/Process/Address" 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:

TRUE	The event state is enabled, events are created.
FALSE	The event state is disabled, no events are created.

relatedProcessVariable:

Address of the Port/Flag

State:

high	Event is generated if assigned Port/Flag changes from 0 to 1
low	Event is generated if assigned Port/Flag changes from 1 to 0
both	Event is generated on every change of assigned Port/Flag

EventName:

Name of the event, generated when the process variable goes from 0 to 1.
Note: There must be an Event Handler defined of the same name.

ParameterList:

List of XML encoded parameter representing the event context. A parameter is written in a single XML- tag with:

```
<ParameterKey _="Value"/>
```

where

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

Example:
Event state configuration with the name 'LowTempBarn12' that creates a 'TemperatureAlert' event when the input port with address MB/IO/I/P4 goes from 0 to 1. The context parameter of the event is defined as Barn: 12 and Temperature < 10.

```
<LowTempBarn12>
      <Enabled _="TRUE"/>
      <ProcessVar _="/Process/MB/IO/I/P4" flank="high"/>
      <Event _="TemperatureAlert">
         <Barn _="12"/>
         <Temperature _="&lt;10"/>
      </Event>
</LowTempBarn12>
```

6.4 Testing

6.4.1 Overview

Testing the configuration can be divided in two parts:

1. Test the creation and the processing of the message jobs.
2. Test the creation of the events.

The first part is the same as described in testing event processing above. You can send an event which in the real application is created by an event state definition with the 'DoOn' command and then check the result.

The second part is described in the following section.

6.4.2 Testing Event Creation

To test the creation of an event the Process subsystem can be set in a Test mode. In this mode the bits of the input ports can be set from the controlling PC by the ProcessTest command and the resulting events can be received as answer of the command. Changes of the input ports by electrical signals are ignored until the Test mode is closed.

There are three modes where the Process subsystem can be:

Run:
This is the normal working mode. When physical input signals changed at the input ports a snapshot of the port state is stored and processed. Additionally every minute processing is carried out (to process single Time commands of the process variables values definition).

Stop:
This mode stops the processing and input ports can be changed without effect.

Test:

This is a special mode similar to “stop" where a single processing cycle of the current snapshot of input port states can be started by the Process command. This command also defines how the processing is done by some flags. Changes of the input ports can be simulated by the Set command inside the Process command . This produces a simulated snapshot. The same snapshot can be set by the system property Set command (see later).

To Test the event creation, proceed as follows:

1. Set the Process subsystem device in the Test mode:

Send: `[<Set _="/Process/Program/Mode" value="Test"/>]`
Tixi Alarm Modem responds: `[<Set/>]`

2. Test the processing by using the ProcessTest command. (see later)

3. Set the Process subsystem back to the Run mode.

Send: `[<Set _="/Process/Program/Mode" value="Run"/>]`
Tixi Alarm Modem responds: `[<Set/>]`

A reset of the Tixi Alarm Modem automatically sets Process Mode to “Run”.

ProcessTest to test the processing of input port changes.

Syntax:

```
<ProcessTest printPV="PrintPVFlags" printEvents="EventsFlags"
genEvents="JobGenerationFlag">
   <Set _="PortAddress" value="Value"/>
         ...
   <Set _="PortAddress" value="Value"/>
</ProcessTest>
```

Description:

This command can be used to test the processing of changes of input ports. The behaviour of the system in the test mode can also be defined.

Parameters:

PrintPVFlags:

n	Do not print process variables **(default)**.
y	Print the name and the state of all process variables after processing.
c	Print the name and the state of all changed process variables after processing only.

EventsFlags:

n	Do not print events **(default)**.
y	Print the names of all events generated by the processing.
d	Print the names and the parameters **(Data)** of all events generated by the processing.

JobGenerationFlag

n	Do not generate jobs **(default)**.
y	Generate jobs.

PortAddress:

see chapter 6.2.1.9.

Value:

State value to set. It depends on the Port Type of the port address:

Port Type	**Range**
I	**0,1**
IB	**0...255**
IW	**0...65.535**
ID	**0...4.294.967.295**

Return:

If no error (command is processed):

```
<ProcessTest>
   XML-ResultPrint
</ProcessTest>
```

XML-ResultPrint:

List of all defined event states where an event state entry is defined by:

printPV ="n" printEvent="n":

```
<EventStateName>
</EventStateName>
```

printPV ="y" or **printPV= "c" printEvent="n":**

```
<EventStateName>
   <PV _="ProcessVariableName" value="ProcessVarValue"
   changed="c"/>
</EventStateName>
```

printPV ="y" or **printPV= "c" printEvent="y":**

```
<EventStateName>
   <PV _="ProcessVariableName" value="ProcessVarValue"
   changed="c" />
   <Event _="EventName"/>
</EventStateName>
```

printPV ="y" or **printPV= "c" printEvent="d":**

```
<EventStateName>
   <PV _="ProcessVariableName" value="ProcessVarValue"
   changed="c" />
   <Event _="EventName">
      <ParameterName _="ParameterValue"/>
   </Event>
</EventStateName>
```

printPV ="n" printEvent="y":

```
<EventStateName>
   <Event _="EventName"/>
</EventStateName>
```

printPV ="n" printEvent="d":

```
<EventStateName>
   <Event _="EventName">
      <ParameterName _="ParameterValue"/>
   </Event>
</EventStateName>
```

If there is an error during event state processing:

```
<EventStateName>
    <ErrNo _="errn"/>
    <ErrTxt _="Error Description"/>
    <Context1 _="ContextValue"/>
    <Context2 _="ContextValue"/>
    <Context3 _="ContextValue"/>
</EventStateName>
```

EventStateName:
Name of the event state processed by the Process subsystem.

ProcessVariableName:
Name of the event variable associated to the event state.

ProcessVarValue:
Value of the process variable associated to the event state.

c:

1	Process variable is changed.
0	Process variable is changed.

EventName:
Name of the event generated by the Event State.

ParameterName:
Name of the parameter of the generated event.

ParameterValue:
Value of the event parameter.

On error (command is not processed):

```
<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	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.

6.5 Access Input Ports and Variables

Alternatively to the ProcessTest, command input ports can be read and set by the Get and Set command because they are part of the system properties. Using system properties you can also refer to port values in the message text.

6.5.1 Read by Get Command

Using TiXML you can read the current state of an input port or variable by the Get command. See chapter 2.4.8.3 for further information.

Get Value

Syntax:
```
<Get _="BitAddress"/>
<Get _="Address"/>
```

Description:
Get the state of the input port or variable addressed by the BitAddress/Address.

Elements:
BitAddress:
see chapter 6.2.1.9.
Address:
see chapter 6.2.1.9.

Examples:
Assume the following state of a 16-bit input port of the module with the address set to 40.

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
0	0	1	0	0	1	0	0	0	0	1	0	0	0	0	0

Get the state of the 3rd bit.
Client sends: **[<Get _="/Process/C40/I/P2"/>]**
Tixi Alarm Modem responds: **[<Get _="1"/>]**

Get the state of the first 8 bits.
Client sends: **[<Get _="/Process/C40/IB/P0"/>]**
Tixi Alarm Modem responds: **[<Get _="36"/>]**

Get the state of the second 8 bits.
Client sends: **[<Get _="/Process/C40/IB/P1"/>]**
Tixi Alarm Modem responds: **[<Get _="4"/>]**

Assume a ProcessVariable with name “Temperature" and value “23" exists.

Get the value of the ProcessVar:
Client sends: **[<Get _="/Process/PV/Temperature"/>]**
Tixi Alarm Modem responds: **[<Get _="23"/>]**

6.5.2 Insert Input Port, ProcessVariable or PLC variable values into the Message Text

You can refer to the current state of the input ports or the value of a ProcessVariable in a message text by inserting an appropriate reference. Because these variables are part of the system state, they are also part of the system properties.

To insert the state of a variable in a message text line use reference to the group. BitAddress/Address see chapter 6.2.1.9.

```
&#xae;BitAddress;
&#xae;Address;
```

Example: Insert the state of the port bit 12 from module with the address set to 40.

Command to write the line:

```
<L _="InputPort12=&a#xae;/Process/C40/I/P12;"/>
```

Resulting line:

```
Input Port 12 = 1
```

Example: Insert the value of the ProcessVariable “Temperature":

Command to write the line:

```
<L _="Temperature=&a#xae;/Process/PV/Temperature;"/>
```

Resulting line:

```
Temperature = 23
```

6.5.3 Set the Input Port by the Set Command

It is also possible to set the input ports using the Set command. This is only possible in the Test Mode of the Process Subsystem. In this case the following ProcessTest command processes the new values. The following steps are required:

1. Set the Process subsystem to the testing mode:
   ```
   [<Set _="/Process/Program/Mode" value="Test"/>]
   ```

2. Set the input ports.
   ```
   [<Set _="/Process/MB/IO/I/P0" value="1"/>]
   [<Set _="/Process/MB/IO/I/P3" value="0"/>]
   ```

3. Start processing with process test without values.
   ```
   [<ProcessTestprintPV="y" printEvents="d" GenEvents="n"
   value="0"/>]
   ```

4. Repeat step 2 and 3 for testing.

5. Set the Process subsystem to the Run mode:
   ```
   [<Set _="/Process/Program/Mode" value="Run"/>]
   ```

6.6 Process Output Ports, Process and PLC Variables

Output ports and independent ProcessVariables can be set by the Set command. The state can also be read by the Get command.
See chapter 2.4.8.4 for further information.

Set Output Port

Syntax:
`<Set _="BitAddress" value="Value"/>`

Description:
Set the state of the output port addressed by the PortAddress to the value.

Elements:
BitAddress:
see chapter 6.2.1.9.
Value:
State value to set. It depends on the Port Type of the port address:

Port Type	**Range**
Q	**0,1**
QB	**0...255**
QW	**0... 65.535**
QD	**0... 4.294.967.295**

Examples:
Delete the 3rd bit of the output port of the module with the address set to 42.
```
    <Set _="/Process/C42/Q/P2" value="0"/>
```

Set the first and the second bit of the output port of the module with the address 40 and delete the bits with the index 2-7.
```
    <Set _="/Process/C40/QB/P0" value="3"/>
```

Set the bit 9 and delete the bits 8, 10-15 of the output ports of the module with the address 42.
```
    <Set _="/Process/C42/QB/P1" value="2"/>
```

Set ProcessVariable

Syntax:
`<Set _="Address" value="Value"/>`

Description:
Set the value of the ProcessVariable addressed by “Address”.

Elements:
Address:
see chapter 6.2.1.9.
Value:
Value to set. (string or number)

Examples:
Set the ProcessVariable “Temperature" to value “23":

```
    <Set _="/Process/PV/Temperature" value="23"/>
```

Set PLC Variable

Syntax:

```
<Set _="Address" value="Value"/>
```

Description:

Set the value of the PLC Variable addressed by “Address”.

Elements:

Address:

see chapter 6.2.1.9.

Value:

Value to set. (string or number)

Examples:

Set the PLC Variable “Temperature" from Device_0 at Bus1 to value “23":

```
   <Set _="/Process/Bus1/Device_0/Temperature" value="23"/>
```

6.7 Variable data types and formats

6.7.1 Variable data types

Following data types are used within Tixi Alarm Modem 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…size characters)
Blob	binary data Array (0...size Byte) currently not supported
Bit	digital value (0...1)
Float	Flaoting point single precision ($\pm 3.402823466*10^{38}$)
Double	Floating point double precision ($\pm 1.7976931348623158*10^{308}$)

6.7.2 Variable data formats

Without any formatting the value of I/Os, process-, system- and PLC variables will be shown native. Tixi Alarm Modem is able to reformat the value into a number format or to replace the status of a boolean variable with a string. The reformatted variable will be used for email generation, data logging and as result of get commands (without own format option).

Example:

```
<Temperature  _="I" ind="1" acc="R" format="R10F+4,2;°C"/>
```

Temperature is an integer value “1234”, but with the format string the output will look like this:

```
<Get _="/Process/Bus1/Device_0/Temperature"/>
<Get _="+12,34°C"/>
```

Formatting output of variables

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 ... format="Elements;Text"/>
```

Description:

The parameter `format` persists of two parts separated by **semicolon**:

1.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 pleasure. 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 values native.

2.part “Text” (option):
Contains a **Text** to be displayed together with the value. The value may be displayed within this text using ist given format of part 1. The position of the value is defined by %%. For some variable additional values (e.g. physical medium and unit) may be included. Second part may be left empty too. In this case no semicolon is necessary.

Example:

Both parts:	“T’F+9,2 ;Radius %% cm“
Only 1. part:	“R16“
Only 2. part:	“; Text with:%% as value“

Format elements (Part 1):

? - logical alternative `?string1,string2`
This command is used to replace the boolean values of a variable by predefined strings. If the variable is not zero string1 is emitted, otherwise string2.

Available for following simpleType values:
Uint8, Uint16, Uint32, Int8, Int16, Int32 mit exp= “0“ und Bit

Example:

```
<Variable _="F" simpleType="Uint8" exp="0" …
format=“?open,closed“/>
```

```
<Get _="/Process/Bus1/Device_0/Variable"/>
```

Tixi Alarm Modem answers:
`<Get _="open"/>` on value 1

*** - case alternative** `*Value1:Text1*Value2:Text2**:Text3`

This command is used to replace a value of a variable by predefined strings. If the variable is equals Value1 Text1 is emitted, if the variable is equals Value2 Text2 is emitted, on every other value Text3 is emitted.

*** separator for values to detect**

**** separator for all other values**

The number of values is not limited.

Available for following simpleType values:

Uint8, Uint16, Uint32, Int8, Int16, Int32 und exp= “0“

Example:

```
<Variable _="R" simpleType="Uint8" exp="0" …
format="*0:low*1:medium*2:high**:faulty"/>

<Get _="/Process/Bus1/Device_0/Variable"/>
```

Tixi Alarm Modem 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)

With upper/lower case the display of hex letters (A-F) is 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 und exp= “0“

Example:

Value lower case:

```
<Variable _="R" simpleType="Uint8" exp="0"... format="r16"/>

<Get _="/Process/Bus1/Device_0/Variable"/>
```

Tixi Alarm Modem answers (variable value=90):

```
<Get _="5A"/>
```

Value upper case:

```
<Variable _="R" simpleType="Uint8" exp="0" ... format="R16"/>

<Get _="/Process/Bus1/Device_0/Variable"/>
```

Tixi Alarm Modem answers (variable value=90):

```
<Get _="5a"/>
```

T – thousand delimiter **T*n***

Defines the thousand delimiter.

n= ,	comma as thousand delimiter (e.g. 12,345,678)
n= .	Punkt als Tausendertrennzeichen (e.g. 12.345.678)
n= `	apostrophe as thousand delimiter (e.g. 12`345`678)
n= empty	no thousand delimiter (default)

Note:
Can be combined with format element „F“ or left alone.

Available for following simpleType values:
Uint8, Uint16, Uint32, Int8, Int16, Int32, Float, Double

<u>Example</u>:

```
<Variable _="R" simpleType="Uint32" exp="0"... format="T. "/>

<Get _="/Process/Bus1/Device_0/Variable"/>
```

Tixi Alarm Modem answers (variable value=98765):

```
<Get _="98.765"/>
```

F - number format

`F Sign padding field width decimal point fixed point numbers`

This command defines the format of the number.
It includes several subitems, which have to be in the given order:

sign: defines, if a sign should emitted

+	the sign is emitted, even if the value is positive (e.g. „+12.3“ , „-12.3“)
-	the sign is only emitted, if the value is negative (e.g. „12.3“ , „-12.3“)
empty	the value is unsigned

padding: defines how empty positions in the output field have to be filled (only used, if output field width is given)

0	the field is filled with zeros (e.g. 0066.3)
empty	unfilled fields are cut off (e.g. 66.3)

field width: gives the maximum size of the number 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: this character is used as decimal separator (option)

,	a comma is used as decimal separator
.	a dot is used as decimal separator (default)

fixed point number: this defines the number of digits behind the decimal separator. Can be omitted, if no decimal point separator is given.

Note:
Can be combined with format element „T“ or left alone.

Available for following simpleType values:
Uint8, Uint16, Uint32, Int8, Int16, Int32, Float, Double

Examples:

The value of the variable is “12345” for all examples:

sign:
```
<Variable _="F" simpleType="Float"… format="F+"/>

<Get _="/Process/Bus1/Device_0/Variable"/>
```
Tixi Alarm Modem answers (Value = 123,45):
```
<Get _="+123.45"/>
```

field width, padding:
```
<Variable _="R" simpleType="Uint32" exp="-3"... format="F09"/>

<Get _="/Process/Bus1/Device_0/Variable"/>
```
Tixi Alarm Modem answers (Value = 123,456):
```
<Get _="00123.456"/>
```

Fix point value , decimal point, fixed point numbers, padding :
```
<Variable _="R" simpleType="Int32" exp="-3"… format="T’F+9.2"/>

<Get _="/Process/Bus1/Device_0/Variable"/>
```
Tixi Alarm Modem answers (Value = 3123,456):
```
<Get _="+3’123.45"/>
```

```
<Variable _="R" simpleType="Int32" exp="0"… format="T’F+9.2"/>

<Get _="/Process/Bus1/Device_0/Variable"/>
```
Tixi Alarm Modem answers (Value = -3123456):
```
<Get _="-312’345"/>
```

Floating pouint valuel, decimal point, fixed point numbers, padding :
```
<Variable _="F" simpleType="Float"… format="T’F+9.2"/>

<Get _="/Process/Bus1/Device_0/Variable"/>
```
Tixi Alarm Modem answers (Value = 3123,456):
```
<Get _="+3’123.45"/>
```

Text (part 2):

%%
Defines the position of the value within output.
This part is available for all types of data. It is the only format option for data type „String“.

Example:

```
<Variable _="R" simpleType="Int32" exp="-2" …
          format="F+;Temp: %%°C"/>

<Get _="/Process/Bus1/Device_0/Variable"/>
```
Tixi Alarm Modem answers (Value = 123,45):
```
<Get _=”Temp: +123.45°C”/>
```

%M% – M-BUS Medium (VIF)
%U% – M-BUS Unit (VIF)
These commands will add the M-BUS Value Information Field data to the value.

Example:

```
<Var01 simpleType="Int32" exp="-2"...
       format=";Medium:%M% value=%%   %U%"/>
<Get _="/Process/Bus1/Device_0/Var01"/>
```

Tixi Alarm Modem answers (Variable value=25,30, Heat counter volume):

```
<Get _="Medium:Heat 0 Volume Flow value=25.30 l/h"/>
```

6.8 Using the service button

On the back of each Tixi Alarm Modem a “service button" can be found. This button can be used to invoke an event.
The configuration is done with a special System-Eventhandler:

Database path: /EVENTS/EventHandler/System

```
<EventHandler>
...
   <System>
      <OnButton>
         <EventHandlerCommands>
         ...
      </OnButton>
   </System>
</EventHandler>
```

“EventHandlerInstruction" ist a list of event handler commands (see chapter 3.8.1) processed as soon as the button is pressed.
Example:

```
<EventHandler>
   <System>
      <OnButton>
        <SendMail _="MessageJobTemplates/ServiceAlarm"/>
      </OnButton>
   </System>
</EventHandler>
```

6.9 Analog input Hutline

The new Tixi Alarm Modem hardware in “Hutline” design offers analog inputs 0-10V (12bit). To convert the value (0-4095, 10V=3798) corresponding to the measured voltage, the modem uses a “periphery” configuration, which is part of the PROCCFG database. Without this configuration the modem (@FW 2.0) automatically converts the values from 0-10000 (10V=10000).

Database path: /PROCCFG/Periphery

Periphery – Analog input

Syntax:

```
<Periphery>
   <Module Name="ADC 1*12bit" Address="Card">
      <Numerator _="Numerator"/>
      <Denominator _="Denominator"/>
      <Tolerance _="Tolerance"/>
      <Rate _="Rate"/>
   </Module>
</Periphery>
```

Description:

Configuration of the analog input used to convert the measured value.

Elements:

Card:

Address of interface or extension card, e.g. analog input on mainboard: “C9a”

Numerator:

Number on top of the fraction to be multiplicated with the measured value.

Denominator:

Number on bottom of the fraction to be multiplicated with the measured value (has to 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. (Default 1000)

Examples:

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 6.2)

OR load the AI into a process variable to define dezimal 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 6.2)
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 6.2)
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>]
```

6.10 S0-Interface

A special Tixi Alarm Modem hardware in “Hutline” design offers two S0-interfaces which are used to count impulses as defined in the S0-interface standard.

Active S0 devices (power supplied interface) has to be connected to “P+/P-“, passiv S0 devices has to be connected to “A+/A-“ (supplied by modem power).

Impulse length specification:
minimum: 250µs up to 250µs*255
maximum: unlimited

The Tixi Alarm Modem counts these impulses into temporary memory. With a special synchronization impulse (internal, external not yet supportet) this value will be written in a readable variable which may be used for data logging or event creation.

The S0 counters and related variables are not written into flash memory. To keep the counted values during power lost, they have to be written into ProcessVars (see chapter 6.2).

The Tixi Alarm Modem offers different S0-interface modes and value convertions which are configured in the periphery group of PROCCFG database.

Database path: /PROCCFG/Periphery

Periphery – S0 interface (@FW 2.0)

Syntax:

```
<Periphery>
   <Module Name="S0 (PIC)" Address="Card">
      <Numerator _="Numerator"/>
      <Denominator _="Denominator"/>
      <Mode _="0xMode"/>
      <Channel0 _="S0-0-length"/>
      <Channel1 _="S0-1-length"/>
      <TimeScale _="Time"/>
   </Module>
</Periphery>
```

Description:

Configuration of the S0-interface mode and value convertion.

Elements:

Card:
Address of interface or extension card, e.g. S0 interface on mainboard: “C3e” or “I3e”

Numerator:
Number on top of the fraction to be multiplicated with the counted impulses.

Denominator:
Number on bottom of the fraction to be multiplicated with the counted impulses (has to be >0).

Mode:
Defines the S0-interface mode using 3 bits: `0xCBA`

Bit C: Defines the synchronization implulse mode (option)

0: synchronization impulse via S0-interface or TimeScale enabled.
1: synchonisation impulse via “S0_Sync” event handler command enabled.

Bit B: Defines operation mode of S0 channel 1

0: relative counter: on synchronization impulse the counted value will be copied into readable variable and channel counter is reset to 0 .

1: absolute counter: on synchronization impulse the counted value will be copied into readable variable (no counter reset to 0).

2: synchonisation impulse interface: channel is used to get synchronization impulse for the other channel (not yet supported)

Bit A: Defines operation mode of S0 channel 0

Modes see Bit B.

Valid mode combinations: 000,100,010,110,001,101,011,111 (012,021 not yet supported)

S0-0-length:
Impulse length (ms) on S0-interface S0-0.

S0-1-length:
Impulse length (ms) on S0-interface S0-1.

Time:
Time between two synchronization impulses (in seconds).

The TimeScale function depends on synchonisation impulse mode:

1. If Bit C is set to 0 and no synchronization impuls interface is defined, Tixi Alarm Modem will generate the synchronization impulse by itself in the given TimeScale beginning from system start.

2. If Bit C is set to 0 and a synchronization impuls interface is defined, Tixi Alarm Modem will use this value to recalculate the counted values:

   Example:
   The external synchronization impuls is expected every 15 minutes. Due to signal interferences the modem receives a “wrong” synchronization impulse after 7 minutes during the 15 minute TimeScale and copies the 300 counted impulses into the radable variable. The Tixi Alarm Modem will now use the TimeScale time to recalculate the counted impulses during this 7 minutes up to a 15 minutes periode using this formula:

   Recalculated Impulses = counted impulses * (TimeScale/MeassuredTime)

   Y = 300 * (15min/7min) = 642 impulses (rounded)

3. If bit C is set to 1, the TimeScale may be ignored (useless).

<u>S0-interface variables:</u>

These variables are created by the system in the process tree under the card address of the S0-interface:

P0:	counted impulses on channel 0 (only if Bit A is <2)
P1:	counted impulses on channel 1 (only if Bit B is <2)
P2:	recalculated impulses on channel 0 using TimeScale (only if Bit B is <2)
P3:	recalculated impulses on channel 0 using TimeScale (only if Bit B is <2)
P4:	measured time of last synchonization periode
P5:	event trigger, set to 1 on synchronization impulse (power off delayed)

P0-P4 are always converted using numerator/denominator.

Examples:

1. Channel0 used for counting (absolute), Channel 1 used for counting (relative). No value convertion. Impulse length 30ms (channel0) and 40ms (channel1). Synchonization impulse created by scheduler event every 5 minutes (300s):

```
<Periphery>
    <Module Name="S0 (PIC)" Address="I3e">
        <Numerator _="1"/>
        <Denominator _="1"/>
        <Mode _="0x100"/>
        <Channel0 _="30"/>
        <Channel1 _="40"/>
        <TimeScale _="300"/>
    </Module>
</Periphery>
```

Values on first cycle, after 300 impulses on both interfaces:

```
P0: 300
P1: 300
P2: ignore
P3: ignore
P4: 300 (fixed, because of scheduler)
P5: 0 (1, if read directly after synchronization impulse)
```

Values on second cycle, after additional 300 impulses on both interfaces:

```
P0: 600
P1: 300
P2: ignore
P3: ignore
P4: 300 (fixed, because of scheduler)
P5: 0 (1, if read directly after synchronization impulse)
```

2. Channel0 used for counting (relative), Channel 1 used for counting (relative). Value convertion (impulse * 11.25). Impulse length 30ms. Synchonization impulse created by TimeScale every 500s after system start:

```
<Periphery>
    <Module Name="S0 (PIC)" Address="I3e">
        <Numerator _="450"/>
        <Denominator _="40"/>
        <Mode _="0x000"/>
        <Channel0 _="30"/>
        <Channel1 _="30"/>
        <TimeScale _="500"/>
    </Module>
</Periphery>
```

Values if 300 impulses counted on channel 0 and 400 impulses on channel 1:

```
P0: 3375
P1: 4500
P2: ignore
P3: ignore
P4: 500 (fixed, because of TimeScale synchonization mode)
P5: 0 (1, if read directly after synchronization impulse)
```

3. For future versions (not yet supported)
   Channel0 used for counting (absolute), Channel 1 used for synchronization impulse (every 900s). No value convertion. 30ms impulse length

```
<Periphery>
      <Module Name="S0 (PIC)" Address="I3e">
            <Numerator _="1"/>
            <Denominator _="1"/>
            <Mode _="0x021"/>
            <Channel0 _="30"/>
            <Channel1 _="30"/>
            <TimeScale _="900"/>
      </Module>
</Periphery>
```

   Values if external synchonization impulse was detected after 900ms and 500 impulses counted on channel0:

```
P0:  500
P1:  ignore
P2:  500
P3:  ignore
P4:  900
P5:  0 (1, if read directly after synchronization impulse)
```

   Values if external synchonization impulse was detected after 400ms and 200 impulses counted on channel0:

```
P0:  200
P1:  ignore
P2:  450
P3:  ignore
P4:  400
P5:  0 (1, if read directly after synchronization impulse)
```

7 Scheduler

The scheduler can be used to create events at predefined times. These events may be status messages, changing variables or changing 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.

Elements:

ScheduleName:

Name of the schedule.

Event:

Name of the event, generated when the schedule time is reached.

ScheduleTimes

List of Attributes describing the times when the event is generated.
(see *times configuration*)

Example:

Scheduler configuration for the "TemperaturStatus" and “TempLog" event.
The TemperaturStatus message will be sent each Monday and Thursday at 19:00 but not in January.
The TempLog writes the current port status every 30 minutes into a logfile.

```
<Schedule>
   <Time1 _="TemperaturStatus">
      <Weekday _="Mo,Th"/>
      <Time _="19:00"/>
      <Month not="Jan"/>
   </Time1>
   <Timer4 _="TempLog">
      <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’s 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"/>

Hour
Description:
Hour inside a day.
Valid attributes: **“0,1,2,3,...,23"**
Example:
Working hours
```
<Hour _=“9-17"/>
```

Time
Description:
Time is used to define 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:
Data is used to define an 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 holiday
```
<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:
```
<Day _=“1-5,25-31"/>
```

Month
Description:
Month valid for all years.
Valid attributes: **“Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec**"
or **“1**"-**“12**".
Example:
```
<Month _=“Feb,6,Oct"/>
```

Weekday

Description:

Weekday valid for all weeks.
Valid attributes: **“Mo, Tu, We, Th, Fr, Sa, Su**"
or **“0" (=Su) – “6" (=Sa)**.

Example:

```
<Weekday _=“Mo-Fr"/>
```

Condition

Description:

Condition is used to refer to other times within /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 to Friday at 8:00-20:00 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 to Friday 7:40 to 16:40 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 New ScheduleDefinition database

To prevent upload problems of the SCHEDULE database groups (conditions for schedulers missing - vice versa) Tixi.Com has made the decision to redesign the SCHEDULE database with introduction of firmware 2.2. We recommend to use the new structure even if the old format is still supported (Note: don't mix old and new structure).
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 and 7.2 for more informations.

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 the scheduler

ScheduleTest *- Gets 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 (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)

Examples:

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 modem 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 Tixi Alarm Modem 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* (@FW 2.0)

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 Logfile group (database LOG)

Example:

Sequencer profile which will call the event “SetProcessVars”. The Profiles will be copied into “Profiles” logfile:

```
<Sequencer>
   <LevelProfile event="SetProcessVars" 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* (@FW 2.0)

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 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 secuencer profiles may be transfered in to 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 enclosured 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;

Return:

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

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

Examples:

Following three examples are in XML format:

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.

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 changes 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>]
```

```
Example 1 in TEXT format:

     <SetSequence _="LevelProfile" priority="0" mode="TEXT" mask="d;t;1;2">
          <![CDATA[
          *.*.*;09:00;20;80
          *.*.*;18:00;30;70
          ]]>
     </SetSequence>
```

To delete a sequence, the sequence definition inside sequencer configuration has to be deleted.

A sequence may also be transferred to the modem via email or Express-Email.
See chapter 9.6 for more informations.

8.2.1 Profile priorities

For each sequencer profile a maximum of 3 priorities is allowed:

The sequenced handles the profiles in different ways, related to the priority:

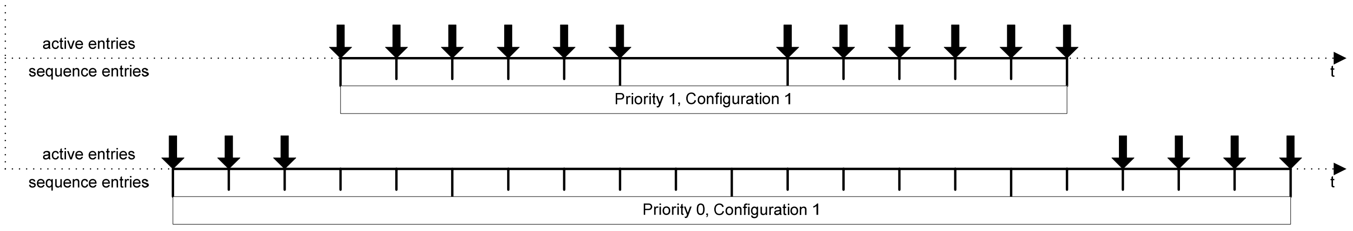
Priority 0:
A sequence with priority 0 will be **replaced** by a sequence with priority 0.

Priority >0
A sequence with priority >0 will be **supplemented** by new data with same priority. All expired entries will be deleted.

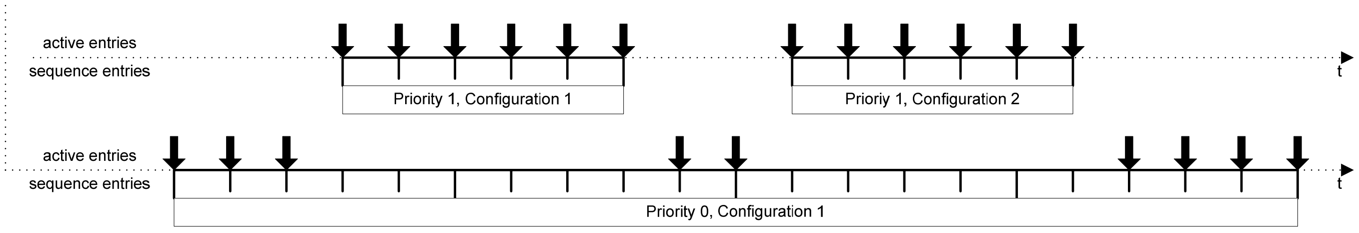
If several sequencer events inside a single profile are configured with the same point of time, the sequence with the highest priority will be processed only.

<u>Examples:</u>

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 the sequencer

***SequenceTest** - Gets a list sequencer event times* (@FW 2.0)

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 (chapter 2.4.4)

Examples:

All sequencer event times of the next 14 days, max 100 entries:

```
<SequenceTest _="LevelProfile" range="next 14 Days"/>
```

All sequencer event times from now until 31.12.2003, max 25 entries:

```
<SequenceTest _="LevelProfile" 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 _="LevelProfile" range="22.8.2003,9:00-2003/10/31,22:00"/>
```

All sequencer event times starting from 1.1.2004, max 50 entries:
```
   <SequenceTest _="LevelProfile" 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>
      <LevelProfile event="SetProcessVars" logfile="Profiles"/>
   </Sequencer>
</SetConfig>]
```

Event Handler to be called by sequencer:

```
<SetProcessVars>
   <Set _="/Process/Bus1/D0/MinValue" value="&#xae;~/P1;"/>
   <Set _="/Process/Bus1/D0/MaxValue" value="&#xae;~/P2;"/>
</SetProcessVars>
```

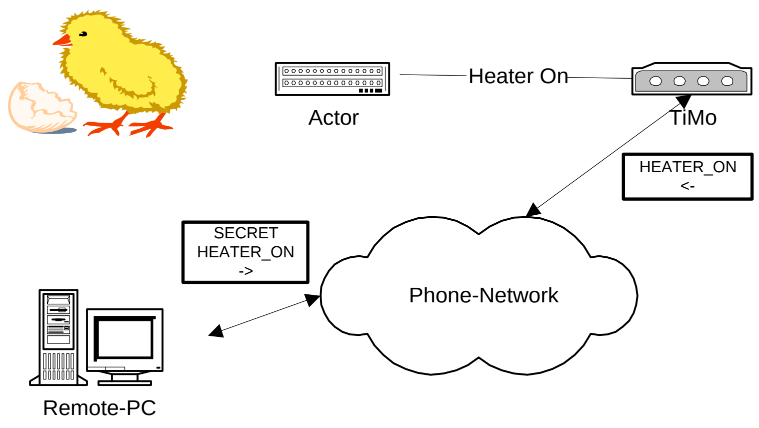
Sequencer times and values:

```
[<SetSequence _="LevelProfile" 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

Tixi Alarm Modem can be controlled by received Express-Emails, E-Mails and SMS or by a simple phone call (callerID). The following picture shows the scenario:



The Tixi Alarm Modem has an extension module with output ports (for example with a module address of C42). At the output ports, some actuators are connected by an electrical signal line (for example a heater).

The remote-PC uses a message including a password (*SECRET*) and a command word (*HEATER_ON*) in the subject line, for example:

SECRET HEATER_ON

Tixi Alarm Modem receives the message and triggers an event with the same name as the command word (in the example 'HEATER_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 message will be converted into upper case, therefore the EventHandler names must be upper case too.

The event is processed according to the commands defined by the corresponding event handler configuration for example:

Database path: /EVENTS/EventHandler

```
<EventHandler>
   <HEATER_ON>
      <Process>
         <LD _="1"/>
         <ST _="MB/IO/Q/P4"/>
      </Process>
      <SendMail _="MessageJobTemplates/AnswerOnHeaterOn"/>
   </HEATER_ON>
</EventHandler>
```

In the example the event sets the port 4 where a heater is connected and started.

Additionally, a job is started which sends an answer message back to the sender of the command such as:

```
OK: HEATER ON
```

9.2 Event via incoming call (callerID)

All Tixi Alarm Modems are able to detect incoming callerIDs using the CLIP service. If a detected callerID matches an entry in the callerID database, the call will not be answered by the Alarm Modem. 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 Alarm Modem.

Elements:

`X:`

Increasing number of entries. Value 1 - 5

`CallerID:`

Transmitted callerID to be detected by the Alarm Modem.
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:

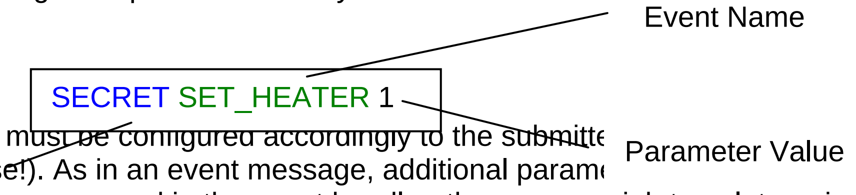
Mobile phone number 01721234567 will activate the event “OpenGarage", mobile phone numbers 01727654321 and 01727654355 will activate the event “OpenDoor”:

```
[<SetConfig _="ISP">
    <IncomingCallTrigger>
        <No1 _="01721234567" event="OpenGarage"/>
        <No2 _="017276543*" event="OpenDoor"/>
    </IncomingCallTrigger>

</SetConfig>]
```

Note: Some providers are transmitting the country code too, e.g. “+491721234567".
In this case you have to this complete number as IncomingCallTrigger.

9.3 Incoming Message Format

To map an incoming message to an event, the message (subject) has to have a special syntax. The following example shows this syntax:



An event handler must be configured accordingly to the submitte MS al se!). As in an event message, additional param se parameters can be processed in the event handler, the message job template or in the message text template.

The following message format is defined.

Incoming Message

<u>Syntax:</u>

`Password SPACE EventName SPACE Parameter1 SPACE Parameter2...`

<u>Description:</u>

Format of the subject line of an incoming message to trigger an event.

<u>Elements:</u>

`Password:`

Password to access the device. May have 1...20 characters (not empty), no SPACE character allowed.

`EventName:`

Name of the event to be triggered (for SMS upper case @FW 2.2). There must be an event handler configured for this event. May have 1...20 characters (not empty), XML tag characters allowed only.

`Parameter1...Parameter10:`

Value of the N-th parameter (no SPACE character allowed, only 29 characters per parameter if using SMS).

<u>Examples</u>:

User password `SECRET`, trigger the `HEATER_ON` event and submit the parameter `1`.

```
SECRET HEATER_ON 1
```

9.4 Events generated by an incoming message

Unlike the default events, the event created by an incoming messages contains predefined parameters. These parameters can be used to control Tixi Alarm Modem or to create an answer message.

Each message type has its own specific parameter list:

9.4.1 Events generated by an incoming Express-E-Mail

Important: Tixi Alarm Modem can only process **uncompressed** Express-E-Mails. If you use a Tixi-Mail Box with “Tixi Server" Application to send an Express-E-Mail to the Alarm Modem, you have to disable the compression manually:

The compression can be disabled in the tixisvr.ini file (located: c:\TixiMail\Tixisvr\tixisvr.ini). Use a text editor (e.g. “Notepad") to edit the file (close Tixi Server first).

Change the line `CompressTixiMail=CompressOn` in the `[TixiMailBox]` section of the file to `CompressTixiMail=CompressOff.`

Unlike the default events, the event created by an incoming Express-E-Mail message contains predefined parameters. These parameters can be used to control Tixi Alarm Modem or to create an answer message.

The event generated by incoming Express-E-Mail message has the following format:

Express-E-Mail Message created Event

Syntax:

```
<DoOn _="EventName">
   <Event _="EventName"/>
   <Password _="Password"/>
   <Alpha _="SenderAlias"/>
   <OA _="OA"/>
   <RemoteSerialNo _="RemoteSerialNo"/>
   <RemoteBoxnumber _="RemoteBoxnumber"/>
   <RemoteBoxname _="RemoteBoxname"/>
   <Time _="ReceiveTime"/>
   <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 Express-E-Mail message described as event command message. Such a message can be created to simulate the receipt of an Express-E-Mail message.

Elements:

EventName:

Event name parsed from received message text.

Password:

Password parsed from received message text.

SenderAlias:

The Express-E-Mail address of the sender.

OA:

Originating address received from telephone network.

RemoteSerialNo:

The serial number of the TixiBox, which sends the message.

RemoteBoxnumber:

The phone number of the sending TixiBox, as defined in the USER database of the sending box (see chapter 3.2).

RemoteBoxname:

The name of the sending TixiBox, as defined in the USER database of the sending box (see chapter 3.2).

ReceiveTime:

Time stamp indicating when the message was received.

MessageText:

The text from the ‘Subject’ line of the received message.

P1...P10(optional)

Value of the parameter delivered by the message if any.

Examples:

Event message generated from an incoming Express-E-Mail of this format: `SECRET HEATER_ON 1`

```
<DoOn ="HEATER_ON">
   <Event _="HEATER_ON"/>
   <Password _="SECRET"/>
   <RemoteSerialNo _="12345"/>
   <RemoteBoxnumber _="+49-30-40608582"/>
   <RemoteBoxname _="Test Tixi Alarm Modem"/>
   <Time _="01/07/20,08:56:33+08"/>
   <Text _="SECRET HEATER_ON 1"/>
   <Alpha _=" TEST+49-30-40608582"/>
   <OA _="03040608582"
   <P1 _="1"/>
</DoOn>
```

In case the Express-E-Mail 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.

Express-EMail Message created Event on event processing error

Syntax:

```
<DoOn _="System/TixiInvalidEvent">
   <Event _="EventName"/>
   <Password _="Password"/>
   <RemoteSerialNo _="RemoteSerialNo"/>
   <RemoteBoxnumber _="RemoteBoxnumber"/>
   <RemoteBoxname _="RemoteBoxname"/>
   <Time _="NetworkTime"/>
   <Text _="MessageText"/>
   <Alpha _="SenderAlias"/>
   <OA _="OA"/>
   <ErrNo _="ErrNo"/>
   <ErrText _="ErrorText"/>
   <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 Express-E-Mail message which could not be processed. This event corresponds to a `DoOn` event message. Note that an event handler of that name must exist.

For testing purposes, such a message can be created manually and used to simulate the receipt of an Express-E-Mail message.

Elements:

EventName:

Event name parsed from received message text.

Password:

Password parsed from received message text.

RemoteSerialNo:

The serial number of the TixiBox which sends the message.

RemoteBoxnumber:

The phone number of the sending TixiBox as defined in the USER database of the sending box (see chapter 3.2).

RemoteBoxname:

The name of the sending TixiBox as defined in the USER database of the sending box (see chapter 3.2).

ReceiveTime:

Time stamp indicating when the message was received.

MessageText:

Received message text.

SenderAlias:

The Express-E-Mail address of the sender.

OA:

Originating address received from telephone network.

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.

Examples:

```
<DoOn _="System/TixiInvalidEvent ">
   <Event _="HEATER_ON"/>
   <Password _="SECRET"/>
   <RemoteSerialNo _="12345"/>
   <RemoteBoxnumber _="+49-30-40608582"/>
   <RemoteBoxname _="Test Tixi Alarm Modem"/>
   <Time _="01/07/20,08:56:33+08"/>
   <Text _="SECRET HEATER_ON 1"/>
   <Alpha _="TEST+49-30-40608582"/>
   <OA _="03040608582"/>
   <ErrNo _="-300"/>
   <ErrText _="Invalid event name"/>
   <P1 _="1"/>
</DoOn>
```

Express-E-Mail Message created Event on invalid password

Syntax:

```
<DoOn _="System/TixiInvalidPassword">
   <Event _="EventName"/>
   <Password _="Password"/>
   <RemoteSerialNo _="RemoteSerialNo"/>
   <RemoteBoxnumber _="RemoteBoxnumber"/>
   <RemoteBoxname _="RemoteBoxname"/>
   <Time _="NetworkTime"/>
   <Text _="MessageText"/>
   <Alpha _="SenderAlias"/>
   <OA _="OA"/>
   <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 Express-E-Mail 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 user is configured in database SMS_Login this system event will always be processed !

For testing purposes, such a message can be created manually and used to simulate the receipt of an Express-E-Mail message.

Elements:

EventName:

Event name parsed from received message text.

Password:

Password parsed from received message text.

RemoteSerialNo:

The serial number of the TixiBox which sends the message.

RemoteBoxnumber:

The phone number of the sending TixiBox as defined in the USER database of the sending box.

RemoteBoxname:

The name of the sending TixiBox as defined in the USER database of the sending box.

ReceiveTime:

Time stamp indicating when the message was received.

MessageText:

Received message text.

SenderAlias:

The Express-E-Mail address of the sender.

OA:

Originating address received from telephone network.

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 Express-E-Mail of this format: `TRY HEATER_ON 1` in case `TRY` is not the correct password:

```
<DoOn _="System/TixiInvalidPassword">
   <Event _="HEATER_ON"/>
   <Password _="TRY"/>
   <RemoteSerialNo _="12345"/>
   <RemoteBoxnumber _="+49-30-40608582"/>
   <RemoteBoxname _="Test Tixi Alarm Modem"/>
   <Time _="01/07/20,08:56:33+08"/>
   <Text _="SECRET HEATER_ON 1"/>
   <Alpha _="TEST+49-30-40608582"/>
   <OA _="03040608582"/>
   <P1 _="1"/>
</DoOn>
```

9.4.1.1 Configuring Express-E-Mail Event Handler

The typical application for the use of incoming Express-E-Mail is assumed to be the following:

1. Log the message.
2. Set an output port.
3. Send an answer Express-E-Mail.

As described above, there are a number of events created automatically by an incoming Express-E-Mail. 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 case events, insert a section as follows into the EventHandler group of the EVENT database. The example assumes that the `AnswerOnHeaterOn` as well as the `AnswerOnError` message jobs have been configured.

Note: The `TixiInvalidPassword` and the `TixiInvalidEvent` sections are contained in a special sub-section SYSTEM of the EVENT database.

Database path: /EVENTS/EventHandler

```
<EventHandler>
   <HEATER_ON>
      <Log _="IncomingMessage">
      <Annotation ="Incoming ExpressEMail "/>
      <Sender _="&#xae;~/Alpha"/>
      <Time _="&#xae;~/Time"/>
      </Log>
      <Process>
         <LD _="1"/>
         <ST _="MB/IO/Q/P4"/>
      </Process>
      <SendMail _="MessageJobTemplates/AnswerOnHeaterOn"/>
   </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/AnswerOnError"/>
      </TixiInvalidEvent>
   </System>
</EventHandler>
```

Event handler for the error cases

Note: If the message(s) created by this event handler should be sent to the sender of the triggering Express-E-Mail, the SenderAlias parameter (Alpha) is only valid for messages which are sent by this specific event handler.

9.4.1.2 Configuring Message Job Templates for the Express-E-Mail answer message

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 is read from the event message generated by the Express-E-Mail. For example.

Database path: /TEMPLATE/MessageJobTemplate

```
<MessageJobTemplates>
   <AnswerOnHeaterOn _="Express-Email">
      <Recipient _="&#xae;~/Alpha"/>
      <Body _="Heater is On"/>
   </AnswerOnHeaterOn>
   <AnswerOnError _="Express-Email">
      <Recipient _="&#xae;~/Alpha"/>
      <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;~/Alpha` characters which are a reference to the `Alpha` parameter provided by the event message.

9.4.2 Events generated by an incoming SMS (GSM and PSTN)[1]

Note: To process incoming SMS with a GSM Alarm Modem the SIM card of the modem must not contain any read or unread SMS.

Several SMS-providers are using different character sets and special characters are converted unexpected. Therefore we recommend to use EventHandler names without special characters.

SMS Message created Event

Syntax:

```
<DoOn _="EventName">
   <Event _="EventName"/>
   <Password _="Password"/>
   <OA _="OA"/>
   <Time _="NetworkTime"/>
   <Text _="MessageText"/>
   <Alpha _="SenderAlias"/>
   <P1 _="ValuesOfParameter1"/>
   <P2 _="ValuesOfParameter2"/>
   <P3 _="ValuesOfParameter3"/>
   <P4 _="ValuesOfParameter4"/>
   <P5 _="ValuesOfParameter5"/>
   <P6 _="ValuesOfParameter6"/>
   <P7 _="ValuesOfParameter7"/>
   <P8 _="ValuesOfParameter8"/>
   <P9 _="ValuesOfParameter9"/>
   <P10 _="ValuesOfParameter10"/>
</DoOn>
```

[1] Not available for Tixi Alarm Modem ISDN (JD/HD).

Description:

Structure of the event created by an incoming SMS message described as event command message. Such a message can be created to simulate the receipt of an SMS message.

Elements:

`EventName:`

Event name parsed from received message text.

`Password:`

Password parsed from received message text.

`OA:`

Originating Address received from GSM network.

`NetworkTime:`

Time stamp received from GSM network.

`MessageText:`

Received message text.

***`SenderAlias`* `(optional):`**

Alphanumerical representation of the originating address if any stored in the chip card.

`P1...P10(optional)`

Value of the parameter delivered by the message if any.

Examples:

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>
```

In case the SMS message cannot be processed properly, a predefined event is triggered that allows notification of the accident to be given and enables the tracking of intrusion attempts.

SMS Message created Event on event processing error

Syntax:

```
<DoOn _="System/SMSInvalidEvent">
   <Event _="EventName"/>
   <Password _="Password"/>
   <OA _="OA"/>
   <Time _="NetworkTime"/>
   <Text _="MessageText"/>
   <Alpha _="SenderAlias"/>
   <ErrNo _="ErrNo"/>
   <ErrText _="ErrorText"/>
   <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 SMS message which could not be processed. This event corresponds to a `DoOn` event message. Note that an event handler of that name must exist.

For testing purposes, such a message can be created manually and used to simulate the receipt of a SMS message.

Elements:

`EventName:`

Event name parsed from received message text.

`Password:`

Password parsed from received message text.

`OA:`

Originating Address received from GSM network.

`NetworkTime:`

Time stamp received from GSM network.

`MessageText:`

Received message text.

`SenderAlias (optional):`

Alphanumerical representation of the originating address if any stored in the chip card.

`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.

Examples:

```
<DoOn _="System/SMSInvalidEvent ">
   <Event _="HEATER_ON"/>
   <Password _="SECRET"/>
   <OA _="+491717959463"/>
   <Time _="01/07/20,08:56:33+08"/>
   <Text _="SECRET HEATER_ON 1"/>
   <Alpha _="CON"/>
   <ErrNo _="-300"/>
   <ErrText _="Invalid event name"/>
   <P1 _="1"/>
</DoOn>
```

SMS Message created Event on invalid password

Syntax:

```
<DoOn _="System/SMSInvalidPassword">
   <Event _="EventName"/>
   <Password _="Password"/>
   <OA _="OA"/>
   <Time _="NetworkTime"/>
   <Text _="MessageText"/>
   <Alpha _="SenderAlias"/>
   <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 SMS 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 user is configured in database SMS_Login this system event will always be processed !

For testing purposes, such a message can be created manually and used to simulate the receipt of an SMS message.

Elements:

EventName:

Event name parsed from received message text.

Password:

Password parsed from received message text.

OA:

Originating address received from GSM network.

NetworkTime:

Time stamp received from GSM network.

MessageText:

Received message text.

***SenderAlias* (optional):**

Alphanumerical representation of the originating address if any stored in the chip card.

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 _="SECRET HEATER_ON 1"/>
   <Alpha _="CON"/>
   <P1 _="1"/>
</DoOn>
```

9.4.2.1 Configuring SMS Event Handler

The typical application for the use of incoming SMS is assumed to be the following:

1. Log the message.
2. Set an output port.
3. Send an answer SMS.

As described above, there are a number of events created automatically by an incoming SMS. 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 case events, insert a section as follows into the EventHandler group of the EVENT database. The example assumes the `AnswerOnHeaterOn` as well as the `AnswerOnError` message jobs to be configured.

Note: The `SMSInvalidPassword` and the `SMSInvalidEvent` sections are contained in a special sub-section `SYSTEM` of the `EVENT` database.

Database path: /EVENTS/EventHandler

```
<EventHandler>
   <HEATER_ON>
      <Log _="IncomingMessage"/>
      <Annotation _="Incoming SMS"/>
      <Sender _="&#xae;~/OA"/>
      <Time _="&#xae;~/Time"/>
      </Log>
      <Process>
         <LD _="1"/>
         <ST _="MB/IO/Q/P4"/>
      </Process>
      <SendMail _="MessageJobTemplates/AnswerOnHeaterOn"/>
   </HEATER_ON>
   <System>
      <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/AnswerOnError"/>
      </SMSInvalidEvent>
   </System>
</EventHandler>
```

Event handler for the error cases

Note: In case the message(s) created by this event handler should be sent to the sender of the triggering SMS, the originating address parameter (`OA`) is valid for messages only which are sent by this specific event handler.

9.4.2.2 Configuring Message Job Templates for the SMS answer message

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 is read from the event message generated by the SMS. For example:

Database path: /TEMPLATE/MessageJobTemplates

```
<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>
```

Note: The sender address is read from the created event. In the Message Job Template, it is represented by the `&#xae;~/OA` characters which are a reference to the `OA` parameter provided by the event message.

9.4.3 Events generated by a received POP3 E-Mail

POP3 E-Mail created Event

Syntax:

```
<DoOn _="EventName">
   <Event _="EventName"/>
   <Password _="Password"/>
   <OA _="SenderAddress"/>
   <Time _="ReceiveTime"/>
   <Text _="MessageText"/>
   <Alpha _="SenderAlias"/>
   <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 a received POP3 e-mail message described as event command message. Such a message can be created to simulate the receipt of a POP3 e-mail message.

Elements:

EventName:

Event name parsed from received message text.

Password:

Password parsed from received message text.

SenderAddress:

Sender address (from field) parsed from received message header.

ReceiveTime:

Time stamp indicating when the message was received.

MessageText:

Received message text.

***SenderAlias* (optional):**

Alias name of sender.

P1...P10(optional)

Value of the parameter delivered by the message if any.

Examples:

Event message generated from a received POP3 email with this subject:
`SECRET HEATER_ON 1`

```
<DoOn ="HEATER_ON">
   <Event _="HEATER_ON"/>
   <Password _="SECRET"/>
   <OA _="support@tixi.com"/>
   <Time _="01/07/20,08:56:33+08"/>
   <Text _="SECRET HEATER_ON 1"/>
   <Alpha _="support@tixi.com"/>
   <P1 _="1"/>
</DoOn>
```

In case the POP3 email message cannot be processed properly, a predefined event is triggered that allows notification of the accident to be given and enables the tracking of intrusion attempts.

POP3 email Message created Event on event processing error

Syntax:

```
<DoOn _="System/POPInvalidEvent">
   <Event _="EventName"/>
   <Password _="Password"/>
   <OA _="SenderAddress"/>
   <Time _="ReceiveTime"/>
   <Text _="MessageText"/>
   <Alpha _="SenderAlias"/>
   <ErrNo _="ErrNo"/>
   <ErrText _="ErrorText"/>
   <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 a received POP3 email message which could not be processed. This event corresponds to a `DoOn` event message. Note that an event handler of that name must exist.

For testing purposes, such a message can be created manually and used to simulate the receipt of a POP3 email message.

Elements:

`EventName:`

Event name parsed from received message text.

`Password:`

Password parsed from received message text.

`SenderAddress:`

Sender address (from field) parsed from received message header.

`ReceiveTime:`

Time stamp indicating when the message was received.

`MessageText:`

Received message text.

`SenderAlias` **`(optional):`**

Alias name of sender.

`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.

Examples:

```
<DoOn _="System/POPInvalidEvent ">
   <Event _="HEATER_ON"/>
   <Password _="SECRET"/>
   <OA _="support@tixi.com"/>
   <Time _="01/07/20,08:56:33+08"/>
   <Text _="SECRET HEATER_ON 1"/>
   <Alpha _="Tixi Support"/>
   <ErrNo _="-300"/>
   <ErrText _="Invalid event name"/>
   <P1 _="1"/>
</DoOn>
```

POP3 email Message created Event on invalid password

Syntax:

```
<DoOn _="System/POPInvalidPassword">
   <Event _="EventName"/>
   <Password _="Password"/>
   <OA _="SenderAddress"/>
   <Time _="ReceiveTime"/>
   <Text _="MessageText"/>
   <Alpha _="SenderAlias"/>
   <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 a received POP3 email 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 user is configured in database SMS_Login this system event will always be processed !

For testing purposes, such a message can be created manually and used to simulate the receipt of a POP3 email message.

Elements:

EventName:

Event name parsed from received message text.

Password:

Password parsed from received message text.

SenderAddress:

Sender address (from field) parsed from received message header.

ReceiveTime:

Time stamp indicating when the message was received.

MessageText:

Received message text.

***SenderAlias* (optional):**

Alias name of sender.

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 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 _="support@tixi.com"/>
   <Time _="01/07/20,08:56:33+08"/>
   <Text _="SECRET HEATER_ON 1"/>
   <Alpha _="Tixi Support"/>
   <P1 _="1"/>
</DoOn>
```

9.4.3.1 Configuring POP3 email Event Handler

The typical application of receive POP3 is assumed to be the following:

1. Log the message.
2. Set an output port.
3. Send an answer email.

As described above, there are a number of events created automatically by a received POP3 email. 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 case events, insert a section as follows into the EventHandler group of the EVENT database. The example assumes the `AnswerOnHeaterOn` as well as the `AnswerOnError` message jobs to be configured.

Note: The `POPInvalidPassword` and the `POPInvalidEvent` sections are contained in a special sub-section `SYSTEM` of the `EVENT` database.

Database path: /EVENTS/EventHandler

```
<EventHandler>
   <HEATER_ON>
      <Log _="IncomingMessage">
      <Annotation _="Received POP3 email"/>
      <Sender _="&#xae;~/OA"/>
      <Time _="&#xae;~/Time"/>
      </Log>
      <Process>
         <LD _="1"/>
         <ST _="MB/IO/Q/P4"/>
      </Process>
      <SendMail _="MessageJobTemplates/AnswerOnHeaterOn"/>
   </HEATER_ON>
   <System>
      <POPInvalidPassword>
         <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/AnswerOnError"/>
      </POPInvalidEvent>
   </System>
</EventHandler>
```

Event handler for the error cases

Note: In case the message(s) created by this event handler should be sent to the sender of the triggering POP3 email, the originating address parameter (`OA`) is valid for messages only which are sent by this specific event handler.

9.4.3.2 Configuring Message Job Templates for the E-Mail answer message

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 is read from the event message generated by the POP3 email. For example:

Database path: /TEMPLATE/MessageJobTemplates

```
<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 which are a reference to the `OA` parameter provided by the event message.

9.4.3.3 Collecting Internet E-Mails

To process incoming POP3 emails, Tixi Alarm Modem has to collect them first.
Make sure that the POP3 details are configured in the ISP database (Chapter 4.7)
This simple EventHandler will do the job:

Database path: /EVENTS/EventHandler

```
<POP3>
   <POP3Query/>
</POP3>
```

The Alarm Modem will only receive messages with matching password.

A good solution can be implemented by combining the POP3 query event with the scheduler. This will collect emails every 15 minutes.

Database path: /SCHEDULE/Scheduler

```
<POP3 _="POP3">
   <Minute _="0,15,30,45"/>
</POP3>
```

9.4.3.4 Filtering E-Mails

The Alarm Modem is able to filter E-Mail messages to shorten online time, skip spam messages and to share a single POP3 account with other modems.
Therefore a user defined filter word can to 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 modem will ignore all messages without this filter word, and even don’t delete them.

9.5 Testing

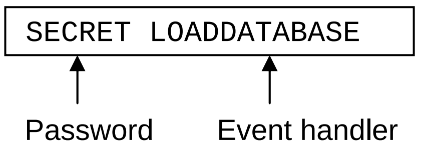
The event handling can be tested independently. Therefore, the event messages created by an incoming message can be "simulated" by sending a DoOn message with the events described above. When this test is finished ok, the message to event map could be tested by sending the corresponding incoming messages.

9.6 Edit databases via incoming messages

Additional to changing variables via incoming messages (chapters above) its possible to replace complete databases e.g. the AddressBook via incoming email or Express-Email. The necessary event handler command “SetConfig” is explained in chapter 3.8.1.

The Tixi Alarm Modem requires a special message syntax, to detect and process the new database content.

At first the subject line of the message has to containt the password (chapter 9.7) and the event handler name with SetConfig command, e.g.:



```
SECRET LOADDATABASE
```

The message body has to containt the databases in following syntax:

Message body syntax – edit databases (@FW 2.0)

Syntax:

```
<D>
   <SetConfig _="DATABASE">
      <Group>
         Data…
      </Group>
   </SetConfig>
</D>
```

Description:

Message body structure to change databases via incoming email or Express-Email.

Elements:

DATABASE:	Name of database to edit, e.g. USER, ISP, PROCCFG. See chapter 12 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 _="TAM-Test@freenet.de"/>
         </MySelf>
         <Receiver>
            <Email _="demo@tixi.com"/>
         </Receiver>
      </AddressBook>
   </SetConfig>
</D>
```

9.7 Configuring Login

To protect the access to the device against an unauthorized control via an incoming message, one or more passwords must be defined. There are two ways to configure a password protection:

- Simple Password.
- Sender data (CallerID, E-Mail Alias) depending.

The simple method defines a password for all senders independently of the sender device used. When using the other variant, the password is valid in conjunction with a specific originating address (callerID) only. For originating address protection (callerID check) with SMS or Express-E-Mail the telephone socket has to support callerID presentation (CLIP). Ask your local telephone company for details.

9.7.1 Simple access rights

The SMS Login (also valid for Express-E-Mail and E-Mail) has to be configured in the USER database.

Database path: /USER/SMS_Login

Remote Control access protection

Syntax:

```
<SMS_Login>
   <Default _="Password"/>
   <OA_nnn _="Password"/>
</SMS_Login>
```

Description:

Enables incoming message access to the Tixi Alarm Modem 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.

Elements:

OA_nnn:

Originating address (callerID) that should be authorized to access the Tixi Alarm Modem. For a simple password protection this address must be set to **Default.** Otherwise, the format is as follows:

OA_*nnn*	Where ***nnn*** is the originating address (callerID) which consists of numbers only (SMS, Express-E-Mail) or letters (email), thus any hyphens and other characters than numbers must be removed. The number is preceded by the characters **OA_**.
Password	The password required for accessing the device. May be empty. Maximum 25 characters.

Example:

Configure a non-sender aware password protection with the password SECRET:

```
[<SetConfig _="USER">
   <SMS_Login>
      <Default _="SECRET"/>
   </SMS_Login>
</SetConfig>]
```

Configure protection for a sender "+49172123456789" and the password SECRET:

```
[<SetConfig _="USER">
   <SMS_Login>
      <OA_49172123456789 _="SECRET"/>
   </SMS_Login>
</SetConfig>]
```

Configure Protection for an E-mail sender with the alias name 'Tixi Support' and the password SECRET:

```
[<SetConfig _="USER">
   <SMS_Login>
      <Tixi_Support _="SECRET"/>
   </SMS_Login>
</SetConfig>]
```

If alias name can not be found within user list, email address will be checked too: Protection for an e-mail sender without alias but with email address Support@tixi.com and the password SECRET:

```
   [<SetConfig _="USER">
      <SMS_Login>
         <Supporttixicom _="SECRET"/>
      </SMS_Login>
</SetConfig>]
```

9.7.2 Advanced Access Rights

See chapter 3.11.2 about advanced access rights. (@FW 2.0)

10 Tixi Alarm Modem and PLC Operation

The Tixi Alarm Modem is not only to be used on it's own, but even in conjunction with PLC devices. As the Tixi Alarm Modem got most common PLC protocols already implemented, connecting it to a PLC is very simple.

Once the connection is established physically via RS232 or RS422/485, the Tixi Alarm Modem configuration may be enhanced by any variable present inside the PLC. The Tixi Alarm Modem will then be able to read the variable values, triggere events thereupon, log the values and send the logfiles. Based on logical instructions or incoming messages, the Tixi Alarm Modem may even set PLC values.

There's a variety of PLC systems to be supported by the Tixi Alarm Modem, which got the respective PLC protocols already implemented. These PLCs may be connected to the Tixi Alarm Modem without any change in programming or configuration, just by means of their RS232 or RS422/485 interface:

- Mitsubishi ALPHA2, MELSEC FX
- Siemens Simatic S7-200, S7-300 (MPI)
- VIPA (GreenCable)
- Moeller Easy 400/500/600/700/800/MFD
- SAIA S-Bus
- Carel PC2
- Modbus
- Tixibus
- M-BUS
- …

Detailed information on configuratioon and usage of Tixi along with PLCs is to be found within the Tixi PLC Manual, which comes along with this package.

11 Appendix: Addresses of serial interfaces and IOs

Alu-Line						
Device Description	Tixi Alarm Modem					
	V.90 / ISDN (Tixi L)	V.90 / ISDN (Tixi XL)	V.90 / ISDN (Tixi XL)	GSM (Tixi XL)	GSM (Tixi XL)	GSM (Tixi XXL)
Product Code	JM20/JD20/JF20	JM20/JD20/JF20 -R153 or R168	JM20/JD20/JF20 -R253 or R268	JMG20-R1-G1	JMG20-R2-G4	JMG20-R368
Housing	KA5 / BB	KA8	KA8	KA8	KA8	KA11
Serial interfaces	1xRS232-F	1xRS232-F, 1xRS232-M	1xRS232-F, 1xRS232-M, 1xRS422/485	1xRS232-F, 1xRS232-M	1xRS232-F, 1xRS232-M or 1xRS422/485 (switchable)	1xRS232-F, 2xRS232-M or 1xRS422/485 (switchable)
I/Os	-	5/3 bzw. 16/8	5/3 bzw. 16/8	2 alternative	2 alternative	2 alternative and 16/8
Systempath:						
RS232-1	COM1 (MB)	COM1 (MB)	COM1 (MB)	COM1 (MB)	COM1 (MB)	COM1 (MB)
RS232-2	-	COM2 (C1)	COM3 (C1)	COM2 (C1)	COM2 (C1) alternative	COM2 (C1) alternative
RS232-3	-	-	-	-	-	COM3 (C3)
RS422/485	-	-	COM2 (C0)	-	COM2 (C1) alternative	COM2 (C1) alternative
Inputs	-	/Process/C0/I/Px	/Process/C0/I/Px	/Process/C0/I/Px	/Process/C0/I/Px	/Process/C0/I/Px (2) and /Process/C2/I/Px (16/8)
Outputs	-	/Process/C0/Q/Px	/Process/C0/Q/Px	/Process/C0/Q/Px	/Process/C0/Q/Px	/Process/C0/Q/Px (2) and /Process/C2/Q/Px (16/8)
GSM-Module	-	-	-	C0	C0	C0

Hut-Line					
Device Description	Tixi Alarm Modem				
	V.90 / GSM / ISDN	V.90 / GSM / ISDN	V.90 / GSM / ISDN	V.90 / GSM / ISDN	V.90 / GSM / ISDN
Product Code	HM10/HD10/HF10/HG20	HM11/HD11/HF11//HG21	HM17/HD17/HF17/HG27	HM41/HD41/HF41/HG41	HM47/HD47/HF47/HG47
Housing	Hut-Line	Hut-Line	Hut-Line	Hut-Line	Hut-Line
Serial interfaces	1xRS232-F	1xRS232-F, 1xRS232-M	1xRS232-F, 1xRS232-M	1xRS232-F, 1xRS422/485	1xRS232-F, 1xRS422/485
I/Os	-	-	2/3 + 10V AI	-	2/3 + 10V AI
Systempath:					
RS232-1	COM1 (MB)	COM1 (MB)	COM1 (MB)	COM1 (MB)	COM1 (MB)
RS232-2	-	COM2 (C0)	-	COM2 (C0)	-
RS232-3	-	-	-	-	-
RS422/485	-	-	COM2 (C0)	-	COM2 (C0)
Inputs	-	-	/Process/MB/IO/I/Px	-	/Process/MB/IO/I/Px
Outputs	-	-	/Process/MB/IO/Q/Px	-	/Process/MB/IO/Q/Px
Analog input	-	-	/Process/MB/A/AI/P0	-	/Process/MB/A/AI/P0
GSM-Module	MB	MB	MB	MB	MB

Hut-Line					
Device Description	Tixi Alarm Modem				
	V.90 / GSM / ISDN	V.90 / GSM / ISDN	V.90 / GSM / ISDN	Ethernet	
Product Code	HM25-2S0 HD25-2S0 HF25-2S0 HG25-2S0	HM71/HD71/HF71/HG71	HM76/HD76/HF76/HG76	HE21/HE27/HE41/HE47	
Housing	Hut-Line	Hutline	Hutline	Hutline	
Serial interfaces	1xRS232-F, 1xRS232-M	1xRS232-F, 1xMPI	1xRS232-F, 1xMPI	1xRS232-F 1xRS232-M (HE21/HE27) 1xRS422/485 (HE41/HE47)	
I/Os	2/1 + 10V AI + 2 S0	-	2/2 + 10V AI	- (HE21/HE41) 2/3 + 10V AI (HE27/HE47)	
Systempath:					
RS232-1	COM1 (MB)	COM1 (MB)	COM1 (MB)	COM1 (MB)	
RS232-2	COM2 (C0)	COM2 (C0)	COM2 (C0)	COM2 (C0) (HE21/HE27)	
RS232-3	-	-	-	-	
RS422/485	-	-	-	COM2 (C0) (HE41/HE47)	
Inputs	/Process/MB/IO/I/Px	-	/Process/MB/IO/I/Px	/Process/MB/IO/I/Px (HE27/HE47)	
Outputs	/Process/MB/IO/Q/Px	-	/Process/MB/IO/Q/Px	/Process/MB/IO/Q/Px (HE27/HE47)	
Analog input	/Process/MB/A/AI/P0	-	/Process/MB/A/AI/P0	/Process/MB/A/AI/P0 (HE27/HE47)	
S0-interface	/Process/I3e/AI/Px	-	-		
GSM-Module	MB	MB	MB	-	

Hut-Line IO-Extensions			
Device Description	Tixi Alarm Modem IO-Extension		
Product Code	XP84D	XP84DR	XP88AD
Housing	Hut-Line	Hut-Line	Hut-Line
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

Notes:
A PLC will be addressed via „auxiliary port“ (interface COM1 (MB) to COMx (Cx)), which has to be part of the process path:

Example.:
Variable of PLC attached to RS232-1 COM1 (MB) with station number „0“:
/Process/AuxMB/D0/Variable

Variable of PLC attached to RS232-2 COM2 (C0) with station number „2“:
/Process/Aux0/D2/Variable (remove “C” from Aux port)

With firmware >=1.80 the interfaces can be addressed as follows (recommended):
MB=COM1, C0=COM2, C1=COM3.
In this case the PLC addressing has to be done via „BusId“ instead of „Aux“ (see PLC TiXML Manual).

12 Appendix: System Properties

This appendix lists the available system properties of the Tixi Alarm Modem 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.

For 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"/>`

Name	Type	Read only	Description
BoxMode	String	yes	Current system mode Modem TiXML
HardwareID	String	yes	Device hardware code
PNP_String	String	yes	Plug and Play string of the device. for example: TIX2010\00238801\MODEM\\Tixi Intelligent Modem ISDN & Fax
FeatureList	String	yes	List of the features (services) of the product. for example: Modem Mode Express E-mail Send Remote Modem-Mode Script Send Job Result Processor
Components	String	yes	List of components of the device. for example: RTC Modem0 FlashOnboard C8
SerialNo	String	yes	Serial Number of the device. for example: 00238801
FreeFileSize	Uint32	yes	Free memory in file system (Bytes)
LocalIPAddr	String	yes	IP address of the modem (assigned during PPP connection)

/EEProm

Name	Type	Read only	Description
GM	String	yes	Interface address of GSM modem, copied from configuration
Pin1	String	Yes	PIN1 for GSM SIM card, copied from configuration
Pin2	String	yes	PIN2 for GSM SIM card, copied from configuration
PD	String	yes	GSM extension I/Os port direction, copied from configuration

/Ethernet

Name	Type	Read only	Description
AssignedIP	String	yes	IP address assigned via BootP or configuration
SubnetMask	String	yes	SubnetMask assigned via BootP or configuration
Link	String	yes	Active link speed of ethernet interface, e.g. 100Mbps
MAC	String	yes	Network interface MAC address

/Firmware

Name	Type	Read only	Description
Version	String	yes	Version number of the firmware. for example: 0.07.00.021
Date	String	Yes	Date of build

/GSM

Name	Type	Read only	Description
Reg	Uint16	yes	Shows network registration state
Reg_Text	String	yes	Verbose network registration state
Operator	String	yes	Name of active GSM operator
Quality	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	Uint16	yes	as RXQUAL values, see ETSI GSM 05.08 (@FW 2.2)
Account	Uint16	yes	Credit left on SIM card (Germany: €€cc)
DaysLeft	Uint16	yes	Days until card expiration (@FW 2.2)

/Hardware/Modules

Name	Type	Read only	Description
RTC	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	String	yes	Type of Modem #0, not defined if not present. for example: JF20-R
Modem1	String	yes	Type of Modem #1, not defined if not present.
GsmModule	String	yes	Type of GSM modem, if present
FlashOnboard	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.
COMx	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
C4x	String	yes	Hutline extension module (C42,C44,C44,C46,C48,C4a,C4c,C4e)

/Hardware/RAM

Name	Type	Read only	Description
Attributes	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	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	Type	Read only	Description
Attributes	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	String	yes	Detected size of the system ROM in bytes, not defined if no information is present.

/Hardware/FileSystem

Name	Type	Read only	Description
Attributes	String	yes	Code describing the attributes of the system RAM, not defined if no information is present. for example: 0
Type	String	yes	Code describing the type of the system RAM, not defined if no information is present.
Size	String	yes	Detected size of the system RAM in bytes, not defined if no information is present. for example: 262144 -> 2 MByte

/LogCounter (dynamic)

Name	Type	Read only	Description
Logfilename	Uint16	no	Number of entries in the specified Logfile. (@FW 2.0) Value can be changed by “set” command, new entries will be added.

/OEM

This sub-tree depends on the product declaration defined by the vendor of the device.

/Process/Program

Name	Type	Read only	Description
Mode	String	no	Processing Mode of the program: Run...Processing is running Stop..Processing is stopped. Test..Testing mode.

/Process/MB

Name	Type	Read only	Description
FirstCycle	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	Uint16	yes	Status of the Service Button.
ModemOffHook	Uint16	yes	State of the modem (@FW 2.0) 0: On hook 1: Off hook (outgoing call) 2: Off hook (incoming call)
TransMode	Uint16	yes	TransMode state: (@FW 2.2) 0: no TransMode established 1: TransMode to COM1 established 2: TransMode to COM2 established
MaxCycleTime	Uint16	yes	Longest cycle time
CycleTime	Uint16	yes	Actual cycle time

/Process/PV (dynamic properties)
Values of the process variables defined by the PROCCFG/ProcessVars

/Process/MB (dynamic properties)
Values of the unnamed process variables defined by the main board (HM,HG,HD,HF).

/Process/Cn (dynamic properties)
Values of the unnamed process variables defined by the installed extension board

/Process/Auxn (dynamic properties)
Values of the process variables defined in PROCCFG/External.

/Process/Busn (dynamic properties) (@FW 2.2)
Values of the process variables defined in PROCCFG/External with BusId.

/TIMES

Name	Type	Read only	Description
TIME	String	yes	Current system time of the day:hour:minutes:seconds
DATE	String	yes	Current system date year/month/day
RFC822Date	String	yes	Current system date and time in the e-mail format according to RFC 822. for example: Fri,13 Jul 01 13:56:00 +0100
PowerOffTime	String	yes	System time of the last power off event with (resolution of 1 minute.) year/month/day/hours/minutes/seconds
PowerOnTime	String	yes	System time of the last power on event with year/month/day/hours/minutes/seconds
DAYOFWEEK	String	yes	Sunday,Monday, …Saturday

DAYOFWEEKNO	String	yes	0-6
YYYY_MM_DD	String	yes	Current system date year_month_day. (@FW 2.2)
HH_MM_SS	String	yes	Current system time of the day hour_minutes_seconds. (@FW 2.2)

/EVENT (Data Base)
See 'EVENT' database description.

/ISP (Data Base)
See 'ISP' database description.

/LOG (Data Base)
See 'LOG' database description.

/PROCCFG (Data Base)
See 'PROCCFG' database description.

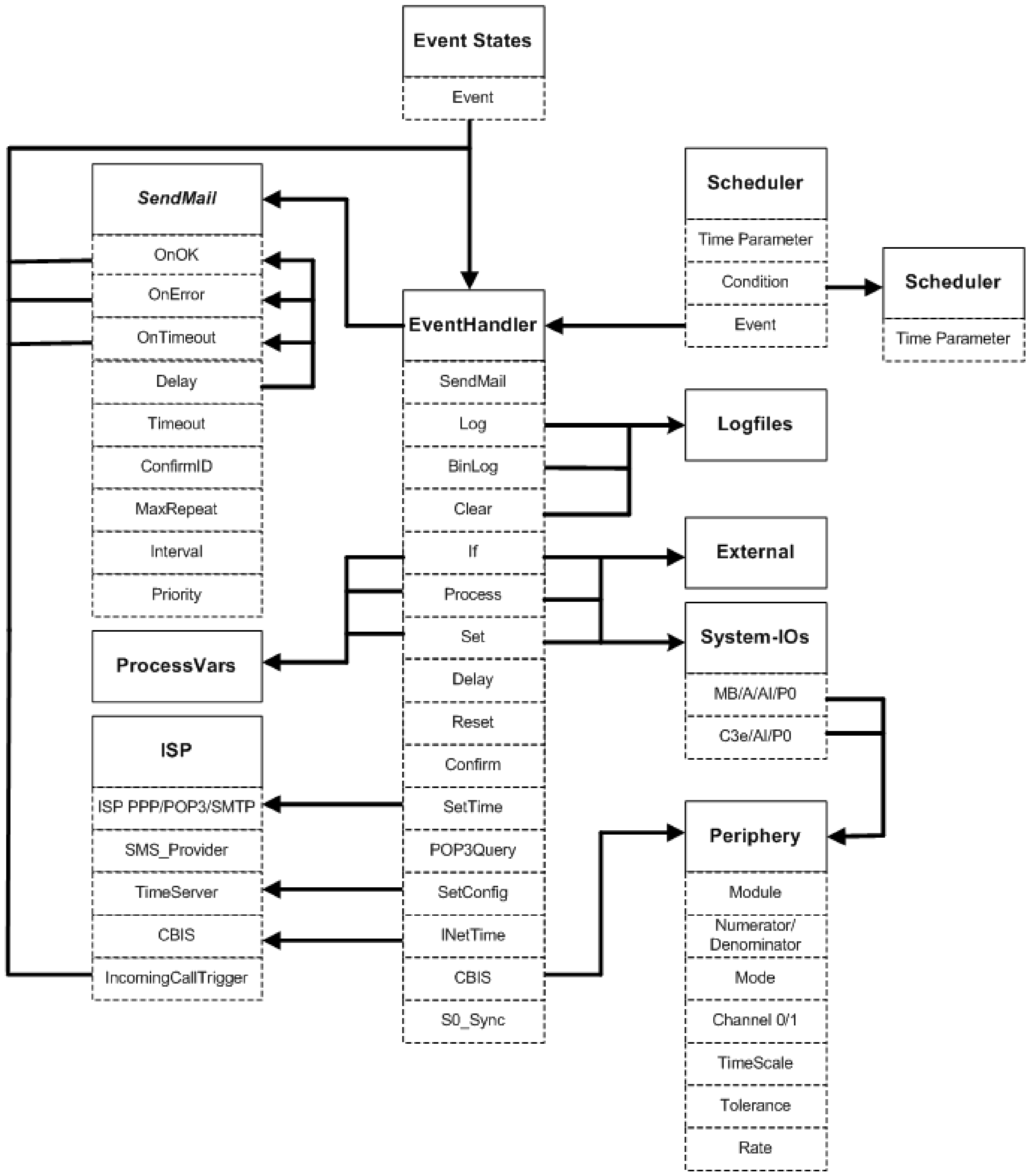
/SCHEDULE (Data Base)
See 'SCHEDULE' database description.

/TEMPLATE (Data Base)
See 'TEMPLATE' database description.

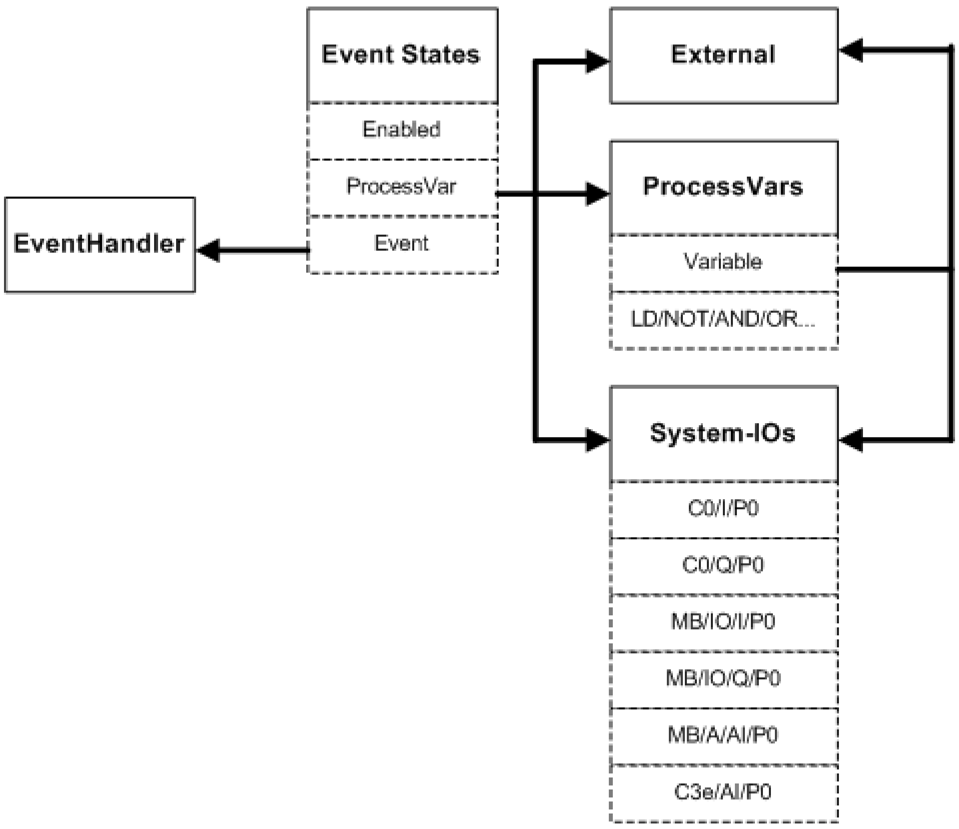
/USER (Data Base)
See 'USER' database description.

13 Appendix: Project structure and connections

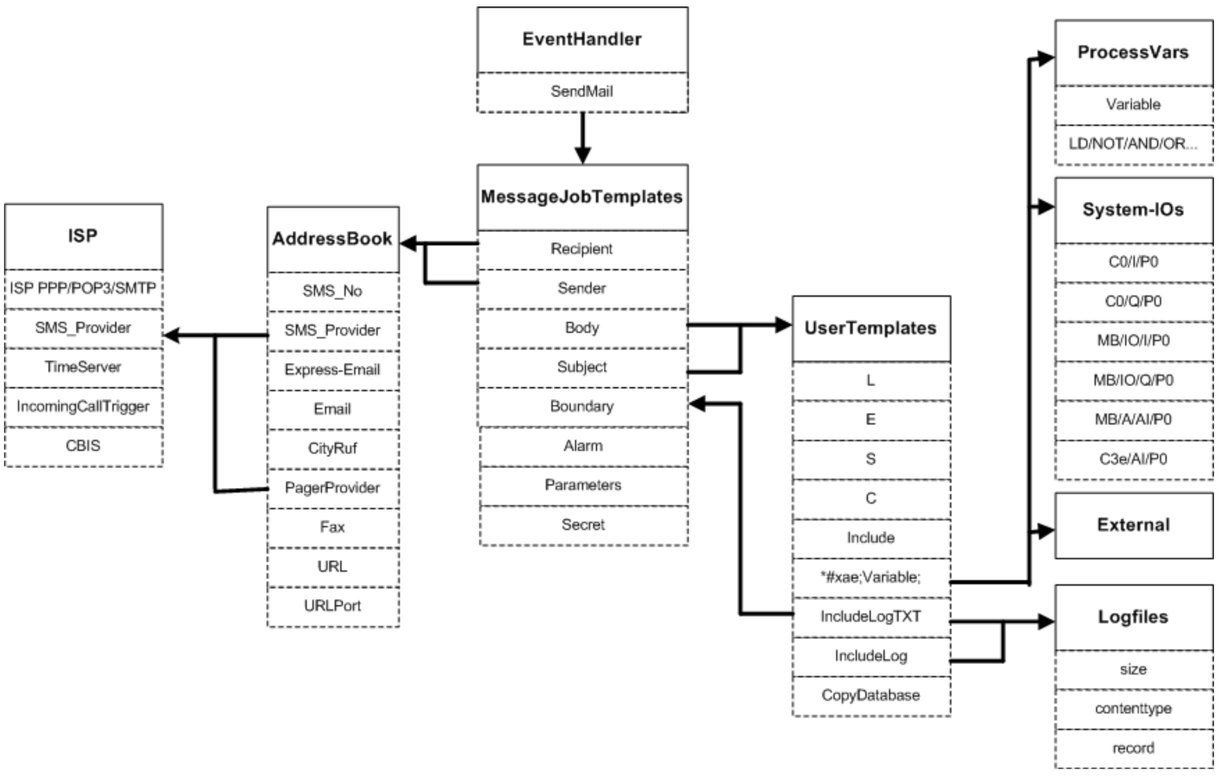
13.1 Event Handler, Scheduler



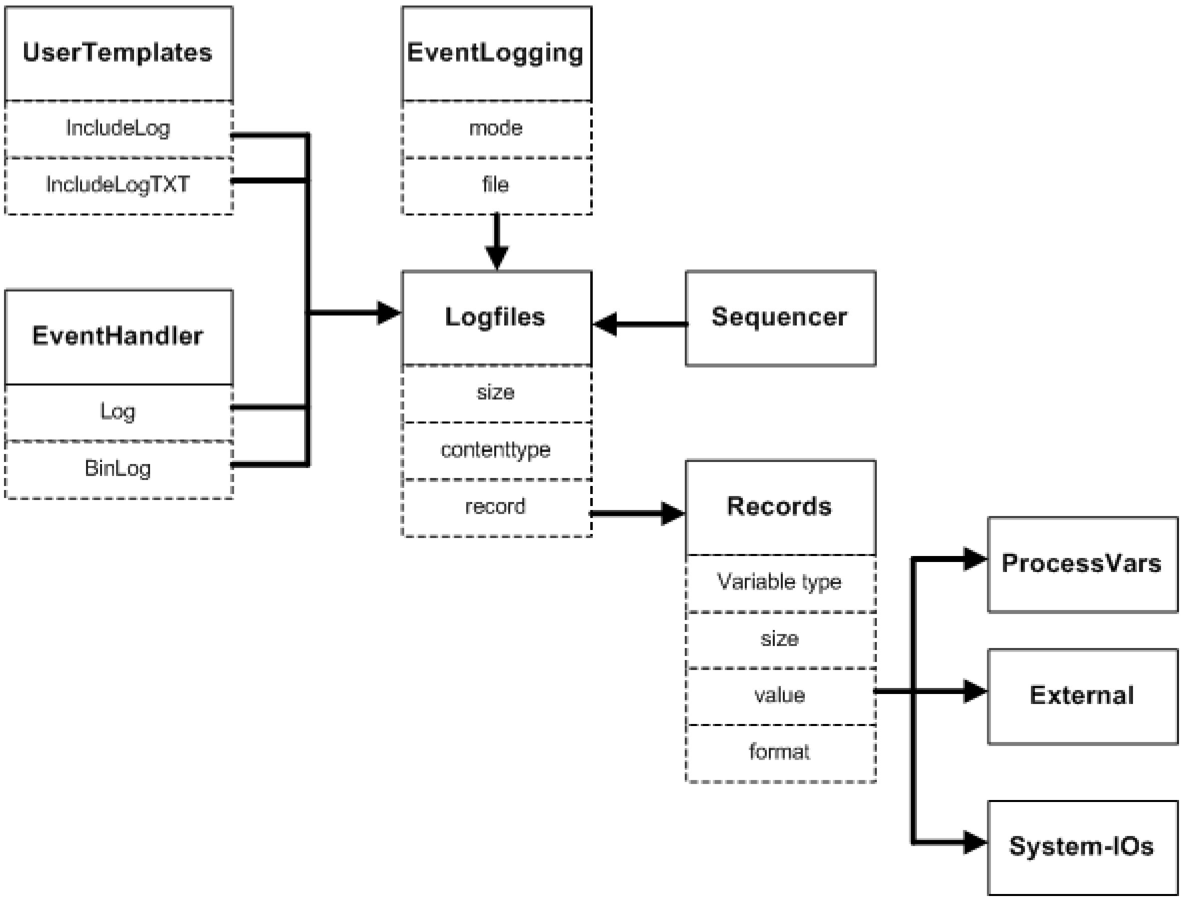
13.2 Event States, External, ProcessVars, System-IOs



13.3 MessageJobTemplates, UserTemplates, AddressBook



13.4 Logfiles, Records, EventLogging



14 Appendix: Firmware

14.1 Compatibility

The Tixi Alarm Modem 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 supports both: the old and the new structure.
Due to this changes a project written for a new firmware may not work with an older firmware.

The following list helps you to determine if project changes are necessary:

Firmware	**Description**
1.52.0.0	New External Format, protocol type (see PLC TiXML manual) A new bus attribute “protocol” now contains the manufacturer and the protocol, the attribute “type” contains just the Master/Slave mode. Old syntax, e.g.: type="sucom,Master" New syntax, e.g.: protocol=”Moeller,Easy 400/600” type=”Master”
1.60.0.184	TIMES-Group (chapter 12) The Tixi Alarm Modem time variables (Date, Time etc.) are moved into a new group called /TIMES. Example: Old path: /Date New path: /TIMES/Date
	AddressBook INet->Email (chapter 3.4) The addressbook parameter for email addresses was renamed. Old name: INet, New name: Email

1.64.0.172	GSMPorts activation in USER-Database (chapter 3.2) The operationg mode of the GSM module IOs can be set via USER-Database entry.
1.63.0.65	MessageJobTemplate Tixi-> Express-Email (chapter 3.7) The MessageJobTemplate type for Exptess-Emails was renamed. Old name: Tixi New name: Express-Email
	Addressbook: TixiMail-> Express-Email (chapter 3.4) The addressbook parameter for Express-Email addresses was renamed. Old name: TixiMail New name: Express-Email
1.63.0.59	Dial Rules T/P:N/Y -> Tone/Pulse,No/WaitForDialTone (chapter 3.3): The dial rule parameters were renamed. Old syntax: T:N, P:N, T:Y, P:Y New syntax: Tone,NoWaitForDialTone, Pulse,NoWaitForDialTone, Tone,WaitForDialTone, Pulse,WaitForDialTone.
1.63.0.51	POP3 Flag: PbS -> POPBeforeSMTP (chapter 3.5) The value name of the POP before SMTP parameter was changed: Old syntax: PbS New syntax: POPBeforeSMTP
	SMTP Flag: d -> DontDelete (chapter 3.5) The value name of the delete message parameter was changed: Old syntax: d New syntax: DontDelete
	AddressBook: SMS_Nr -> SMS_No (chapter 3.4) The addressbook parameter for SMS numbers was renamed. Old name: SMS_Nr New name: SMS_No
1.72.0.0	Handshake SUCOM -> noDTR The handshake parameter on Bus configuration and TransMode for Moeller Easy and Mitsubishi Alpha XL was renamed. Old name: SUCOM New name: noDTR
1.80.86.0	Time-Server moved from database ISP to ISP/ISP PPP-Server moved from database ISP/ISP to ISP CBIS moved from database ISP/ISP to ISP
1.81.0.0	HG GSM modem definition in USER-DB: old <GSMModem _="MB"/> (no longer allowed) new <GSMModem _="MB"/>
2.00.0.0	New AccRights database. Database USER/Login, USER/SMS_Login no longer supported. New LogDefinition database. Database LOG/Logfiles, LOG/Records no longer supported.
2.1.25.0	Incoming SMS command names will be transformed to upper case, therefore all EventHandler for SMS remote switching have to be upper case.
2.01.36.0	S0-Interface changed from "C3e" to "I3e"
2.1.39.0	External Bus parameter "mem" now specified in "bytes" (previously KB)
2.02.1.0	New ScheduleDefinition database. Database SCHEDULE/Schedule, SCHEDULE/Condition no longer supported.
2.2.8.0	Logfile template CSV now without quotes
2.2.21.0	Tixi bus variable "S" (String) now requires "size" instead of "length"

14.2 Feature History

Firmware	**Description**
1.51.0.0	New PLC Support: S7-200, VIPA, Mitsubishi FX + Alpha XL, TixiBus (see PLC TiXML Manual) D_ON, D_OFF (ON/OFF-Delay) (see chapter 6.2.1.6)
1.52.0.0	IncomingCallTrigger (see chapter 9.2), Delete Active Jobs command
1.52.2.11	Support for Moeller Easy Byte-, Word- , DWord Markers (see PLC TiXML Manual)
1.52.2.50	GSM Supports Data calls on Voice number, Support GSM I/O ports, Support Mobilcom Austria SMS, POP3 remote switching alias check (see chapter 9.7)
1.62.0.13	Binary data logging (see chapter 4)
1.63.0.4	EventHandler Delay command (see chapter 3.8.1)

1.63.0.50	Clear Log command (see chapter 4.8)
1.64.0.172	factory reset keeps GSM settings
1.64.0.28	Internet time synchonization (see chapter 3.12)
1.66.0.1	EventStates Enabled 0/1 (see chapter 6.3)
1.66.0.184	Define Webserver HTTP-Port
1.70.1.1	List of ProcessVars with Values
1.70.12.0	Clear RS232 send buffer after DTR low
1.70.13.0	GSM Account Query (see chapters 3.2, 12)
1.70.15.0	Show GSM state at Line-LED
1.70.8.0	GSM Informations: Registration, Signal-Quality, State (see chapter 12)
1.70.9.0	Reject remote command: Switch ModemMode
1.72.0.0	**MAJOR RELEASE (Aluline)** DAYTIME format string (see chapter 3.13), SMS receive enable/disable, Modem OffHook State System variable (see chapter 12), GSM Info Operator, Reg-TXT (see chapter 12)
1.80.17.0	Generate S0 sync pulse
1.80.6.0	SetTime Event Handler command (see chapter 3.8.1)
1.80.7.0	Support for two bus protocols on two different COM ports (see PLC TiXML Manuel)
1.80.x	Support for COM1, COM2 interface names (see chapter 11)
1.80.22.0	Supoort for ASCII protocol
1.80.41.0	Call Back Initiation Service CBIS
1.80.61.0	Support for authenticated SMTP with CRAM-MD5
1.80.71.0	Processing quoted printable email format
1.80.72.0	TFTP file transfer
1.80.85.0	New access rights database
1.80.89.0	Sending base64 attachments
1.80.94.0	Logfile format templates (CSV, XML, HTML)
1.81.1.0	SMS Provider number format national/canonical
1.81.4.0	TiXML via TCP/IP
1.81.7.0	HTTP cache, HTTP restricted access, HTTP Logfile formats
1.81.9.0	PLC slave device state
1.81.11.0	Event handler commands: clear, reset
1.81.15.0	GSM-Firmware in Systemproperties
1.81.16.0	New record types: byte, word, dword, float, double…
1.81.19.0	Modbus RTU integer variables, UseCache, Webserver SSI ReadLog
2.0.0.0	**MAJOR RELEASE (Hutline)** New ProcessVar commands: GTI, LTI, GEI, LEI. Easier syntax, e.g. <NOT _=””/> now <NOT/>
2.0.1.0	IF condition for EventHandler, TransMode Systemproperty
2.0.4.0	SupportLog
2.0.5.0	New ProcessVar commands (math operations, strings), more flexible handling of instructions. CallerID-Support for UK
2.0.9.1	FIND_BIT instruction
2.0.10.0	Easy 500/700 Support
2.1.2.0	S7 MB, MW, MD support
2.1.14.0	TiXML Handshake
2.1.15.0	S7-MPI protocol and TS-Adapter hardware
2.1.19.0	Multiline SMS template
2.1.21.0	Cancel local transmode by PnP detection
2.1.22.0	New Time variables YYYY_MM_DD and HH_MM_SS
2.1.25.0	Incoming SMS command names will be changed to upper case
2.1.27.0	Names for Log columns TiXML Readlog format CSV
2.1.29.0	Logging with “path”
2.1.30.0	Variable format redesigned AddInfo parameter for PLC variable errors
2.1.31.0	Auto TransMode Log Event trigger
2.1.34.0	GSM account query string GSM account expiry GSM V.110 ISP dialup
2.1.35.0	Log all phone calls
2.1.36.0	“Set Value” formats: binary, hex, octal Support A1 account query
2.1.46.0	Confirm “*”
2.1.47.0	Short dial numbers (dialing without location)

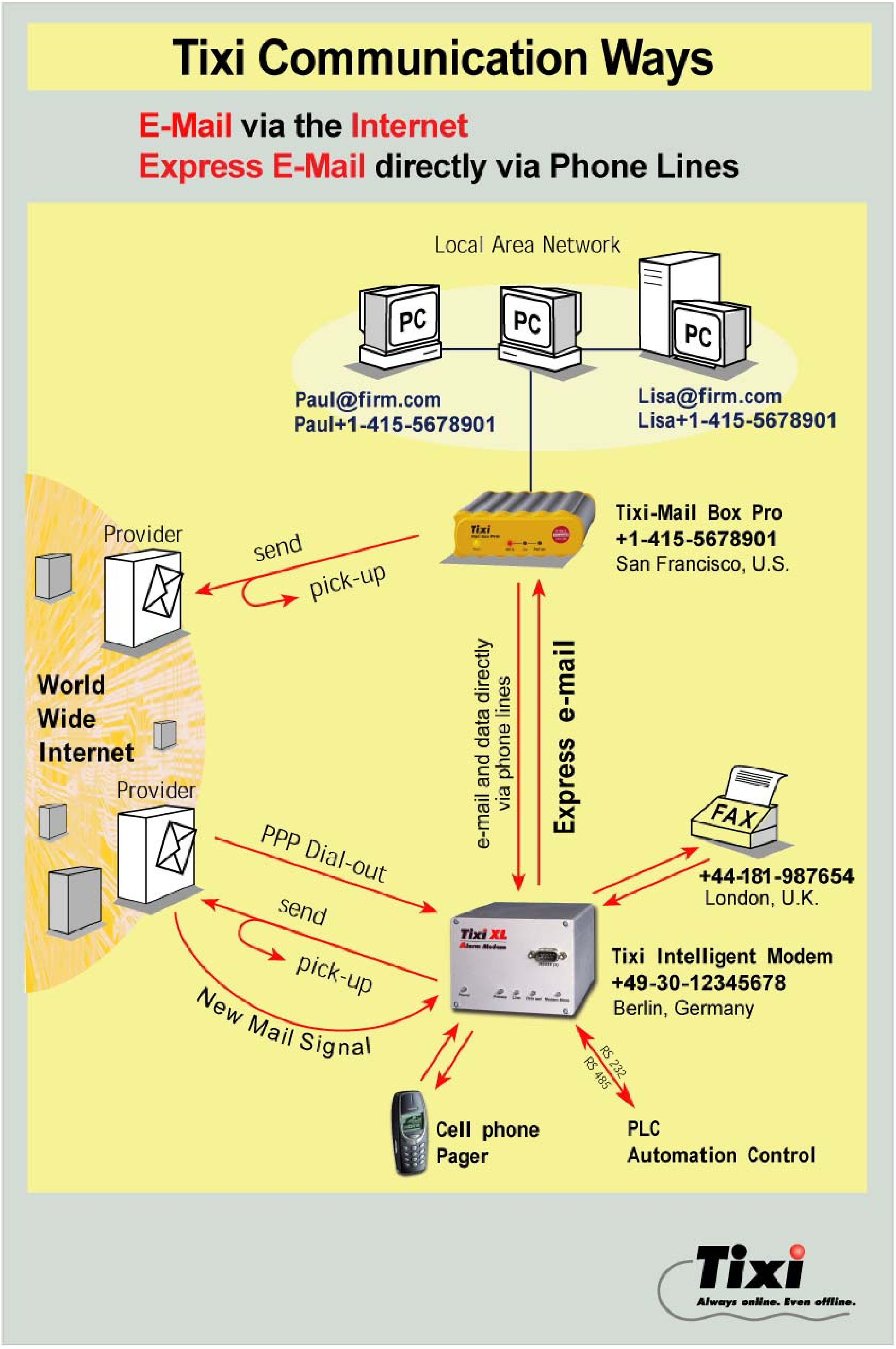
2.1.52.0	Internet (POP3/SMTP/CBIS/InetTime) via Ethernet POP3 Filter
2.1.56.0	EventHandler “Switch” command
2.1.77.0	CBIS email with IP as link
2.2.1.0	GSM BitErrorRate ScheduleDefinition ESMTP PLAIN authentication
2.2.12.0	**MAJOR RELEASE (Aluline and Hutline)**
2.2.19.0	ESMTP plain
2.2.29.0	M-Bus variables for manufacturer, primary- and secondary address
2.2.41.0	M-Bus reset (application codes)
2.2.60.0	EventHandler commands for TransMode
2.2.72.0	SMTP HELO host name
2.2.74.0	**MAJOR RELEASE (Hutline)** New MPI implementation

15 Appendix: References

[1] www.w3.org/TR/SOAP or msdn.microsoft.com/soap
[2] www.w3.org/XML

Index

Notes


Tixi Communication Ways
E-Mail via the Internet
Express E-Mail directly via Phone Lines
Local Area Network
PC
PC
PC
Paul@firm.com
Paul+1-415-5678901
Lisa@firm.com
Lisa+1-415-5678901
Tixi-Mail Box Pro
+1-415-5678901
San Francisco, U.S.
Provider
send
pick-up
World Wide Internet
Provider
e-mail and data directly via phone lines
Express e-mail
FAX
+44-181-987654
London, U.K.
PPP Dial-out
send
pick-up
New Mail Signal
Tixi XL
Tixi Intelligent Modem
+49-30-12345678
Berlin, Germany
RS 232
RS 485
Cell phone
Pager
PLC
Automation Control
Tixi
Always online. Even offline.