see and get more…

ASMEN *- Transmission Drivers and Protocols*

User’s Manual

Doc. No. ENP4001
Version: 27-07-2006

ASKOM

ASKOM Sp. z o. o., ul. Józefa Sowińskiego 13, 44-121 Gliwice,
tel. +48 (0) 32 3018100, fax +48 (0) 32 3018101,
http://www.askom.com.pl, e-mail: office@askom.com.pl

Table of Contents

1. Drivers and Transmission Protocols

The **asix** system includes a set of drivers that handle the following types of data transfer with controllers of an industrial process.

Driver	**Protocol**
ADAM	- protocol for ADAM-4000 modules of ADVANTECH
AGGREGATE	- the driver allows definition of variables, values of which are generated as a result of calculations performed on other variables of the **asix** system (source variables)
CtAK	- the AK protocol allows data exchange between **asix** system computers and Emerson MLT2 analyzers
AM_SA85	- protocol for communication with the Modbus Plus network of Schneider Automation
AS511	- protocol using programmer interface of SIMATIC PLCs of SIEMENS
AS512	- protocol of CP524/525 communication processors for SIMATIC PLCs of SIEMENS
AS512S7	- protocol of CP340 communication processors for SIMATIC S7 PLCs of SIEMENS
BAZA	- the driver enables data import from databases to the **asix** system
BUFOR	- general purpose protocol for information exchange with user programs by means of a shared memory
CALEC MCP	- Calec MCP driver retrieving the current values of variables from CALEC MCP devices of Aquametro according to the protocol described in the document „MCP Datenauslesung mit dem lowlevel Protokoll"
CAN_AC_PCI	- protocol for data exchange between SELECONTROL MAS PLCs of Selectron Lyss AG and **asix** system computers
CANOPEN	- CANOPEN network protocol of SELECTRON MAS PLCs of Selectron Lyss AG
COMLI	- protocol (COMunication Link) for communication with ABB SattCon, AC 800C, AC 800M, AC 250 PLCs. Data are transferred via RS-232 or RS-485 serial interfaces
DATAPAF	- protocol for connection with DataPAF energy counters
DDE	- driver defining a channel of the ASMEN module referring to variables shared by a DDE server
DP	- protocol for devices compatible with PROFIBUS DP by using a PROFIboard card
DP5412	- protocol for devices compatible with PROFIBUS DP by using Siemens cards
DMS285	- protocol for emission meter D-MS285 computers
DMS500	- protocol for emission meter D-MS500 computers (previous name – DURAG)
DSC	- protocol for data exchange between **asix** system computers and DSC 2000 controllers
DXF351	- protocol for communication with Compart DXF351 devices of Endress+Hauser

CtEcoMUZ	- protocol for data exchange between the **asix** system and Microprocessor Protecting ecoMUZ Devices made by JM Tronik
FESTO	- protocol using a diagnostic interface for FESTO PLCs
FILE2ASIX	- the driver enables data import from text files to the **asix** system
FP1001	- protocol for water and steam flow monitors of METRONIC Kraków
GFCAN	- protocol of CAN network with use of communication card of Garz & Fricke Industrieautomation GmbH
K3N	- protocol for data exchange between K3N meters family of OMRON and **asix** system computers
K-BUS	- protocol K-BUS used for data exchange between VIESSMANN Dekamatic boilers controllers connected to a Dekatel-G (or Vitocom 200) concentrator and **asix** system computers
CtLG	- protocol of LG Master-K and Glofa GM PLCs
LUMBUS	- protocol for data exchange between RG72 controllers manufactured by Lubuskie Zakłady Aparatów Elektrycznych (Electrical Measuring Instrument Works) "LUMEL" in Zielona Góra and **asix** system computers
MACMAT	- GAZ-MODEM protocol used for communication with MACMAT stations
M-BUS	- subset of standard protocol for data reading from measuring devices used by MULTICAL heat meters of KAMSTRUP A/S
MEC	- protocol of data exchange between **asix** system and MEC07 and MEC08 heat meters manufactured by Instytut Techniki Cieplnej (Institute of Thermal Technology) in Lodz. Data are transferred with use of a standard RS-232 interface.
MELSECA	- protocol of A1SJ71C24-R2 communication processor for MELSEC-A PLCs
MEVAS	- protocol for data exchange between the MEVAS emission meter computer produced by Lubuskie Zakłady Aparatów Elektrycznych (Electrical Measuring Instrument Works) "LUMEL" in Zielona Góra and **asix** system computers
MODBUS	- subset of standard communication protocol used by AEG Modicon GE Fanuc PLCs
MODBUS_TCPIP	- protocol of data exchange between **asix** system and computers/devices by means of the MODBUS protocol on the basis of Ethernet with the TCP/IP protocol
MODBUSSLV	- MODBUS protocol, in which **asix** operates as SLAVE
MPI	- protocol of MPI interface of SIMATIC S7 PLCs of SIEMENS; a serial interface
MPS	- serial interface protocol for MPS measuring gauges of a power network from OBR Metrologii Elektrycznej in Zielona Góra
MSP1X	- protocol MSP1X used for data exchange between MSP1X PLCs of ELMONTEX and **asix** system computers
MUPASZ	- hollow (virtual) channel protocol
MUZ	- protocol for data exchange between Microprocessor Security Devices of MUZ-RO type
NONE	- NONE protocol enables: - **asix** application testing in simulation mode,

	- data exchange between **asix** programs by means of process variables;
OMRON	- enables data exchange between OMRON PLCs and **asix** system computers
OPC	- the driver defining a channel of ASMEN module retrieving the variables shared by an OPC server
PPI	- protocol for SIEMENS S7-200 PLCs
PROTHERM	- driver for data exchange between Protherm 300 DIFF PLCs of Process-Electronic GmbH and **asix** system computers
PROTRONICPS	- PROTRONIC PS protocol of Hartmann & Braun
S700	- protocol S700 used for data exchange between Maihak 3700 gas analyzers and **asix** system computers
SAPIS7	- protocol of SIMATIC S7 PLCs with use of the MPI interface or PROFIBUS communication processor (an implementation of S7 function)
S-BUS	- S-BUS protocol used for data exchange between PCD PLCs of SAIA Burgess Electronics and **asix** system computers
CtSbusTcpip	- driver for data exchange between family of PCD SAIA-Burgess PLCs and **asix** system computers
SINECH1	- protocol of CP1430 communication processors of SIMATIC S5 PLCs (Ethernet)
SINECL2	- protocol of CP5430 communication processors of SIMATIC S5 PLCs of SIEMENS
SNPX	- driver for data exchange between **asix** system computers and GE Fanuc 90-30 PLCs as well as GE Fanuc 90 CMM and PCM modules
SPA	- protocol used for communication with devices connected to SPA bus of the ABB company
SRTP	- driver used for data exchange between the **asix** system and GE Fanuc Automation VersaMax Nano/Micro PPLCs using an IC200SET001 converter and WersaMax 90 PLCs using the IC693CMM321 communication module; via Ethernet with the TCP/IP protocol
TALAS	- protocol of TALAS emission computers
CtTwinCAT	- driver for data exchange between the **asix** system and the TwinCAT system of Beckhoff Industrie Elektronik
ZDARZENIE ZMIENNA	- driver for generating process variables of WORD type (16-bit word) on the basis of actual values of alarm events in the **asix** system
CtZxD400	- driver of protocol of electric energy counters of ZxD400 type manufactured by Landys & Gyr

The package of available protocols will be systematically expanded. The ASKOM company is ready to develop on customer request any transmission protocol according to the rules defined in the price list of the **asix** system.

1.1. Driver of ADAM Protocol

Driver Use

The ADAM driver is used for data exchange with ADAM-4000 series modules developed by Advantech. The transmission is performed with use of serial interfaces via standard serial ports of a computer (using the converter) or by using an additional card with an RS485 interface.

The only **asix** system requirement is that the ADAM modules should be configured to the following data transfer mode:

- number of character bits 10 (1 start bit, 8 character bits, 1 stop bit),
- no parity check,
- checksum.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel which operates according to the ADAM protocol is given below:

logical_name=ADAM,no,type,port,[bauds]

where:

no	- network number of the ADAM module;
type	- identifier of the ADAM module type. At present the following types are implemented: 1 - ADAM-4011 2 - ADAM-4012 3 - ADAM-4013 4 - ADAM-4017 5 - ADAM-4018 6 - 8-channel pulse counter Mcom-1 (an equivalent of ADAM-4080D) 7 - ADAM-4050 8 - ADAM-4052 9 - ADAM-4060 10 - ADAM-4053 11 - ADAM-4080 12 - ADAM-4021
port	- serial port name;
bauds	- transmission speed.

The *bauds* parameter is an optional parameter. Its default value is 9600 (Bd).

EXAMPLE

An exemplary item which declares the use of transmission channel operating according to the ADAM protocol is given below:

```
CHAN1=ADAM,1,5,COM1,9600
```

The transmission channel of the logical name CHAN1 has the following parameters:

- ADAM protocol;
- network number 1;
- module type - ADAM-4018;
- transmission speed 9600 Bd.

Addressing the Process Variables

The syntax of symbolic address which is used for process variables belonging to the ADAM driver channel is as follows:

VARIABLE_TYPE variable_index [.subchannel_no]

where:

VARIABLE_TYPE	- string identifying the variable name in the ADAM protocol;
variable_index	- variable index within a given type;
subchannel_no	- subchannel number for multichannel modules or a bit number for digital in/out modules.

The following symbols of process variable types are allowed:

R	- read only variable;
W	- write only variable;
RW	- read/write variable.

Depending on the ADAM module type, various ranges of *variable_index* and *subchannel_no* are allowed. Process variables implemented at present are given below:

Table 1. Types of Implemented Process Variables Serviced by ADAM Modules.

Symb. Address	Variable Type in Device	Type of Raw Variable	Device Type	Data Format
	Variables Only for Reading			
R1	Read analog Input	Float	ADAM-4011	Eng. units
R1	Read analog Input	Float	ADAM-4012	Eng. units
R1	Read analog Input	Float	ADAM-4013	Eng. units
R1	Current readback	Float	ADAM-4021	Eng. Units
R1.0	Read analog Input 0	Float	ADAM-4017	Eng. units
R1.1	Read analog Input 1	Float	ADAM-4017	Eng. units
R1.2	Read analog Input 2	Float	ADAM-4017	Eng. units
R1.3	Read analog Input 3	Float	ADAM-4017	Eng. units
R1.4	Read analog Input 4	Float	ADAM-4017	Eng. units
R1.5	Read analog Input 5	Float	ADAM-4017	Eng. units
R1.6	Read analog Input 6	Float	ADAM-4017	Eng. units
R1.7	Read analog Input 7	Float	ADAM-4017	Eng. units
R1.0	Read analog Input 0	Float	ADAM-4018	Eng. units
R1.1	Read analog Input 1	Float	ADAM-4018	Eng. units
R1.2	Read analog Input 2	Float	ADAM-4018	Eng. units
R1.3	Read analog Input 3	Float	ADAM-4018	Eng. units
R1.4	Read analog Input 4	Float	ADAM-4018	Eng. units
R1.5	Read analog Input 5	Float	ADAM-4018	Eng. units
R1.6	Read analog Input 6	Float	ADAM-4018	Eng. units
R1.7	Read analog Input 7	Float	ADAM-4018	Eng. units
R1.0	Read counter/frequency value channel 0	Dword	MCom-1	Hex
R1.1	Read counter/frequency value channel 1	Dword	MCom-1	Hex
R1.2	Read counter/frequency value channel 2	Dword	MCom-1	Hex
R1.3	Read counter/frequency value channel 3	Dword	MCom-1	Hex
R1.4	Read counter/frequency value channel 4	Dword	MCom-1	Hex
R1.5	Read counter/frequency value channel 5	Dword	MCom-1	Hex
R1.6	Read counter/frequency value channel 6	Dword	MCom-1	Hex
R1.7	Read counter/frequency value channel 7	Dword	MCom-1	Hex
R2.0	Read timer interval value channel 0	Dword	MCom-1	0.1 sec incr.
R2.1	Read timer interval value channel 1	Dword	MCom-1	0.1 sec incr.
R2.2	Read timer interval value channel 2	Dword	MCom-1	0.1 sec incr.
R2.3	Read timer interval value channel 3	Dword	MCom-1	0.1 sec incr.
R2.4	Read timer interval value channel 4	Dword	MCom-1	0.1 sec incr.
R2.5	Read timer interval value channel 5	Dword	MCom-1	0.1 sec incr.
R2.6	Read timer interval value channel 6	Dword	MCom-1	0.1 sec incr.
R2.7	Read timer interval value channel 7	Dword	MCom-1	0.1 sec incr.

Table 2. Types of Implemented Process Variables Serviced by ADAM Modules (continuation).

Symb. Address	Variable Type in Device	Type of Raw Variable	Device Type	Data Format
R1.0	Read digital Input 0	Word	ADAM-4050	0/1
R1.1	Read digital Input 1	Word	ADAM-4050	0/1
R1.2	Read digital Input 2	Word	ADAM-4050	0/1
R1.3	Read digital Input 3	Word	ADAM-4050	0/1
R1.4	Read digital Input 4	Word	ADAM-4050	0/1
R1.5	Read digital Input 5	Word	ADAM-4050	0/1
R1.6	Read digital Input 6	Word	ADAM-4050	0/1
RW1	Analog Data Out/Last value readback	Float	ADAM-4021	Eng. Units
RW1.0	Read/Write digital Output 0	Word	ADAM-4050	0/1
RW1.1	Read/Write digital Output 1	Word	ADAM-4050	0/1
RW1.2	Read/Write digital Output 2	Word	ADAM-4050	0/1
RW1.3	Read/Write digital Output 3	Word	ADAM-4050	0/1
RW1.4	Read/Write digital Output 4	Word	ADAM-4050	0/1
RW1.5	Read/Write digital Output 5	Word	ADAM-4050	0/1
RW1.6	Read/Write digital Output 6	Word	ADAM-4050	0/1
RW1.7	Read/Write digital Output 7	Word	ADAM-4050	0/1
R1.0	Read digital Input 0	Word	ADAM-4052	0/1
R1.1	Read digital Input 1	Word	ADAM-4052	0/1
R1.2	Read digital Input 2	Word	ADAM-4052	0/1
R1.3	Read digital Input 3	Word	ADAM-4052	0/1
R1.4	Read digital Input 4	Word	ADAM-4052	0/1
R1.5	Read digital Input 5	Word	ADAM-4052	0/1
R1.6	Read digital Input 6	Word	ADAM-4052	0/1
R1.7	Read digital Input 7	Word	ADAM-4052	0/1
RW1.0	Read/Write digital Output 0	Word	ADAM-4060	0/1
RW1.1	Read/Write digital Output 1	Word	ADAM-4060	0/1
RW1.2	Read/Write digital Output 2	Word	ADAM-4060	0/1
RW1.3	Read/Write digital Output 3	Word	ADAM-4060	0/1
R1.0	Read counter/frequency value - channel 0	Dword	ADAM-4080	Hex
R1.1	Read counter/frequency value - channel 1	Dword	ADAM-4080	Hex
RW1.0	Read/Write initial counter - channel 0	Dword	ADAM-4080	Hex
RW1.1	Read/Write initial counter - channel 1	Dword	ADAM-4080	Hex
W1.0	Clear counter - channel 0	Dword	ADAM-4080	Hex
W1.1	Clear counter - channel 1	Dword	ADAM-4080	Hex

Table 3. Types of Implemented Process Variables Serviced by ADAM Modules (continuation).

Symb. Address	Variable Type in Device	Type of Raw Variable	Device Type	Data Format
R1.0	Read digital Input 0	Word	ADAM-4053	0/1
R1.1	Read digital Input 1	Word	ADAM-4053	0/1
R1.2	Read digital Input 2	Word	ADAM-4053	0/1
R1.3	Read digital Input 3	Word	ADAM-4053	0/1
R1.4	Read digital Input 4	Word	ADAM-4053	0/1
R1.5	Read digital Input 5	Word	ADAM-4053	0/1
R1.6	Read digital Input 6	Word	ADAM-4053	0/1
R1.7	Read digital Input 7	Word	ADAM-4053	0/1
R1.8	Read digital Input 8	Word	ADAM-4053	0/1
R1.9	Read digital Input 9	Word	ADAM-4053	0/1
R1.10	Read digital Input 10	Word	ADAM-4053	0/1
R1.11	Read digital Input 11	Word	ADAM-4053	0/1
R1.12	Read digital Input 12	Word	ADAM-4053	0/1
R1.13	Read digital Input 13	Word	ADAM-4053	0/1
R1.14	Read digital Input 14	Word	ADAM-4053	0/1
R1.15	Read digital Input 15	Word	ADAM-4053	0/1

The ADAM driver is installed as a DLL automatically.

Driver Configuration

The driver configuration is defined in the **[ADAM]** section, which allows to configure the driver for data exchange with ADAM series 4000 modules.

☑ ***DEFAULT_REPLAY_LENGHT=YES|NO***

Meaning	**-** using the default answer length; YES – waiting for the maximal possible answer length or character timeout; NO – it is used if the answer length is not known in order not to wait for a timeout by the answer; otherwise the default answer length is used and a character timeout is waited.
Default value	- NO.

☑ ***CHECKSUM =YES|NO***

Meaning	**-** using the checksum in transfers PC <--> ADAM.
Default value	- YES.

☑ ***READ_TIMEOUT = number***

Meaning	**-** timeout for answer as a multiple of 100 milliseconds.
Default value	- 15, i.e. 1500 ms.

☑ ***CHAR_TIMEOUT = number***

Meaning	**-** character timeout as a multiple of 10 ms.
Default value	- 5, i.e. 50 ms.

1.2. Aggregate Driver

Driver Use

The Aggregate driver allows definition of variables the values of which are generated as a result of calculations performed on other variables of the **asix** system (source variables). Archive values of source variables are used for an aggregate calculation. Usage of archive values allows to prevent any discontinuities in case of the **asix** system restart.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel operating with the Aggregate driver is as follows:

Channel_Name = AGGREGATE

where:

Channel_Name	**-** channel name in the [ASMEN] section.

Addressing the Process Variables

The address part of variable declaration for the Aggregate driver takes the following form:

aggregate_name aggregate_parameters

where:

aggregate_name	- name of the aggregate;
aggregate_parameters	- aggregate parameters, delimited with white space.

The driver may realize following aggregates:

Table 4. Types of Aggregates Executed by the Driver Aggregate.

Aggregate Name	Way of Calculations
Average (Average)	As a result, the weighted average of the source variable in the calculation period is obtained.
Max (max)	As a result, the maximum value of the source variable in the calculation period is obtained
Min (min)	As a result, the minimum value of the source variable in the calculation period is obtained.

Parameters of above aggregates take the following form:

Variable_name:Archive_Type Period Threshold [A] [L[*lower_limit*]:[*upper_limit*]]

where:

Variable_name	- name of the source variable which is connected with the aggregate;
Archive_type	- one letter code determining the type of the archive in which source variable values are saved;
Period	- calculation period of the aggregate;
Threshold	- minimum number of correct measures, in percentages, needed for aggregate calculations;
A	- parameter determines whether calculation performing time should be adjusted to the calculation period;
Lower_limit	- lower limit value; if the source variable value is lower than the value of *lower_limit,* then the *lower_limit* value is used instead; the parameter may be used up to version 1.01.000 of the driver;
Upper_limit	- upper limit value; if the source variable value is higher than the value of *upper_limit*, the then *upper_limit* value is used instead; the parameter may be used up to version 1.01.000 of the driver;

Period determines the calculation period. The calculation period is given in the same manner as the time interval specification for ASPAD, ie. in the form of numbers and units:

<number><unit> [*<number><unit>* [...]]

where:

<number>	- number of given subsequently time units;	
<unit>	- determines the time unit which may be:	
	s	- second,
	m	- minute,
	g lub h	- hour,
	d	- day (24 hours) .

In case when unit is missing, the minute is taken as a default unit of the calculation period.

The result of aggregate calculations is said to be good if the interest rate of correctly read samples (given in percentages) is equal to *Threshold*. Default value of *Period* is 5 minutes, and of *Threshold* - 80 percentages. For the *threshold* correct measurement calculation the source variable valid time is taken into account, according to the parameterization of archiving this variable (sampling period). It means, *threshold* is calculated as a ratio of the sum of all correctness times of measurements and calculation period length. In case of an average, the calculation result is a weighted average in relation to the measurement correctness time. Values of variables, for which time stamp is greater or equal to beginning of the calculation period and lower than the end of it, are taken into consideration for calculations of aggregates. The aggregate calculation occurs after the end of the calculation period.

The last, optional parameter *A* determines the time instance at which an aggregate will be calculated. If the parameter is omitted, the aggregate will be calculated after each reading of the source variable (in stepwise manner). If *A* is the last parameter, then the aggregate calculation time is adjusted to a multiplicity of the aggregate calculation period. The aggregate calculation result type is adjusted to the type of conversion function given in the variable declaration.

EXAMPLE

An exemplary variable declaration:

Temp_sr, Temp-średnia, SREDNIA Temperatura:B 10 70 A, Srednie, 1, 1, NOTHING_FP

The variable `Temperatura_sr` declared above is an average value of the variable `Temperatura`. The period, over which the variable was averaged, is 10 minutes and in order to obtain a correct value of the average, at least 70 percent of correct measurements are needed. Archive values, placed in the B archive, of the variable `Temperatura` are used for the average creation. The aggregate calculation instant will be adjusted to the calculation period multiplicity, i.e. calculations will be performed at 00:00:00, 00:10:00, 00:20:00 and so on.

1.3. CtAK - Driver of AK Protocol for Emerson MLT2 Analyzers

Driver Use

The driver of the AK protocol allows data exchange between **asix** system computers and Emerson MLT2 analyzers. Communication is realized over the RS-485 serial links.

The driver allows only to read the data available by the READ type command, except commands assigned to service purposes as well as commands with the CODE type attribute.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel operating with the CtAK driver is as follows:

Channel=UNIDRIVER, CtAK, *Port=number [;Baudrate=number] [;RecvTimeout] [;CharTimeout]*

where:

UNIDRIVER	- universal driver name;
CtAK	- name of the driver used for communication with a PLC;
Port	- number of a serial port COM;
Baudrate	- transmission speed between a computer and a device; the following speeds are available: 300, 600,1200,2400, 4800, 9600, 19200 Bd; by default - 19200 Bd;
RecvTimeout	- timeout (w milliseconds) between sending the last character of a query and receiving the first character of a response; by default - 200 milliseconds;
CharTimeout	- timeout (w milliseconds) between the successive response characters; by default - 30 milliseconds;

NOTICE *The parameters passed in the channel declaration have to be compatible with the parameters set for communication ports of PLCs handled by this channel.*

EXAMPLE

An exemplary declaration of the channel with standard timeouts on COM2:

CHANNEL = UNIDRIVER, CtAK, Port=2

Addressing Process Variables

The syntax of symbolic address which is used for the variables belonging to the CtAK driver channel is presented below:

<Type>.<AnalNo>.<ChanNo>[.RangeNo].<Index>

where:

Type	- variable type – a name of the command (CODE) used by the protocol for reading particular categories of variables from the analyzer are used as *Type*;
AnalNo	- analyzer number;
ChanNo	- channel analyzer number;
RangeNo	- *Range* number of a given channel (if it is used when addressing the variable - see: statement of commands);
Index	- specific interpretation of *Index* for the *Type* parameter is as follows: a/ number of an element in *Range* (if *Range* is used when addressing the variable - see: statement of commands), b/ number of elements in the data array returned by the command (if the command returns a data array). In particular the array dimension can equal 1.

```
/* concentration: analyzer no.  1, channel; 2,  index 1
JJ_01, concentration,              AIKO.1.2.1,   CHANNEL, 1, 1, NOTHING_FP
/* pressure: analyzer no. 1, channel 3,  index 1
JJ_02, pressure,                   ADRU.1.3.1,   CHANNEL, 1, 1, NOTHING_FP
```

Input/Output States and Calibration State

The driver allows to read Input/Output states of DIO cards by the service command „ASVC Kn S615 b". The following variable address should be used for reading Input/Output states:

ASVC.*<AnalNo>*.*<ChanNo><DIONo>*.*<Index>*

where:

AnalNo	- analyzer number,
ChanNo	- channel analyzer number,
DIONo	- number of a DIO card the calibration status is transferred by,
Index	- offset in the array of data read form the DIO card. The following values are allowable:

1 – states 8 inputs I1 ... I8	(I1 – the least significant bit)
2 – states 8 inputs O1 ... O8	(O1 – the least significant bit)
3 – states 8 inputs O9 ... O16	(O9 – the least significant bit)
4 – states 8 inputs O17 ... O24	(O17 – the least significant bit)

The O24 output state includes the current calibration status.

EXAMPLE

Examples for reading input/output states:

```
# analyzer no. 1, channel 2, DIO no. 3, input state I1 ... I8
JJ_01, state I1...8,    ASVC.1.2.3.1,   PT3, 1, 1, NOTHING

# analyzer no. 3, channel 1, DIO 4,  output state O1 ... O24
JJ_02, state O1...24, ASVC.3.1.4.2,   PT3, 1, 1, NOTHING_DW
```

State of Transmission with the Analyzer

The driver stores information on the status of the last transmission session and on the value of the *Error Code* field transferred from the analyzer during the last transmission (if the transmission ended properly). Reading this information is realized by the following symbolic addresses:

IERR.< *Analnr>.<Channr>*

ICOM.< *Analnr >*

where:

Analnr	- analyzer number,
Channr	- number of the analyzer channel.

IERR returns the value of the *Error Code* field of the last transmission session in a given channel of the analyzer.

ICOM returns the status of the last transmission session with a given analyzer (0 - correct, 1 – transmission error).

Driver Configuration

The driver configuration is performed by using a separate section named [CTAK]. By means of this section it is possible to declare:

- log file and its size,
- log of telegrams,

LOG_FILE=file_name

Meaning	- the item allows to define a file which all the diagnostic messages of the driver will be written to. The item is used for diagnostic purpose.
Default value	- by default, the log file is not created.

LOG_FILE_SIZE=number

Meaning	- the item allows to define the size of the log file.
Default value	- by default, the log file is not created.
Parameter:	
number	- size of the log file In MB.

☑ ***LOG_OF_TELEGRAMS=YES/NO***

Meaning	- the item allows to write the content of telegrams sent/received by the driver to the log file (declared by means of the item LOG_FILE). Writing the content of telegrams should be used only while the **asix** start-up.
Default value	- NO; by default, the driver does not write the content of telegrams to the log file.

EXAMPLE

An exemplary driver parameterization section:

```
[CTAK]
LOG_FILE=d:\tmp\ctAk\ak.log
LOG_FILE_SIZE=3
LOG_OF_TELEGRAMS=YES
```

Statement of Commands

There are implemented data types (commands of the AK protocol) and parameters used when addressing variables of the particular types below.

Table 5. List of Implemented Commands of the AK Protocol for the CtAK Driver.

Type (command)	**Addressing by *Range***
AAEG	Yes
AALI	Yes
AANG	Yes
AEMB	No
AGRW	No
AIKG	No
AIKO	No
AKAK	Yes
AKAL	Yes
AKON	???
ALCH	Yes
ALIK	No
ALIN	Yes
ALKO	Yes
ALST	No
AM90	No
AMBA	Yes
AMBE	Yes
AMBU	Yes
AMDR	No
AQEF	No
ASOL	No
ASTF	Nie
ASTZ	No
ASYZ	No
AT90	No
ATEM	Nie
ATOL	No
AVEZ	No
Options	
ABST	No
AKEN	No
AKOW	Yes
AUKA	Yes
ASVC	Yes (DIO number is used instead of *Range*)

1.4. AM_SA85 - Driver of MODBUS PLUS Protocol for AM-SA85-000 Card

Driver Use

The AM_SA85 driver is used for data exchange between the Modbus Plus network of Schneider Automation and **asix** system computers provided with the AM-SA85-000 card of Schneider Automation.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel operating according to the AM_SA85 protocol is given below:

logical_channel_name=AM_SA85*, address [,adapter [,discrete [,registers]]]*

where:

AM_SA85	- driver name;
Address	- address in the Modbus Plus network in the ASCII string form of R1.R2.R3.R4.R5 format, where R1 ... R5 represent sequent levels of routing to device in the network;
adapter	- adapter number;
discrete	- max. number of discrete values read in an individual telegram;
registers	- max. number of registers read in an individual telegram.

Default values:

adapter	- 0,
discrete	- 120,
registers	- 120.

EXAMPLE

The definition of logical name CHANNEL operating according to the AM_SA85 protocol and exchanging data with the controller no. 10 (other parameters are default ones):

```
CHANNEL=AM_SA85, 10.0.0.0.0
```

The AM_SA85 driver is loaded as a DLL automatically.

Addressing the Process Variables

The symbolic address syntax is compatible to the way of addressing used for the MODBUS protocol.

Driver Configuration

The AM_SA85 driver may be configured by using the **[AM_SA85]** section, which is placed in the initialization file of an **asix** application. Individual parameters are passed in separate items of the section. Each item has the following syntax:

item_name=[number [,number]] [YES] [NO]

☑ ***LOG_FILE=file_name***

Meaning	- the item allows to define a file to which all the diagnostic messages of the AM_SA85 driver and the information about the content of telegrams received by the drive will be written. If the item doesn't define a full path, the log file will be created in the current directory. The log file should be used only during the **asix** system start-up.
Default value	*- by default, the file log isn't created.*

EXAMPLE

```
LOG_FILE=D:\asix\AM_SA85.LOG
```

☑ ***LOG_FILE_SIZE = file_size***

Meaning	- it allows to determine a log file size in MB.
Default value	- by default, 1 MB.

☑ ***LOG_OF_TELEGRAMS=YES|NO***

Meaning	- the item allows writing to the log file (declared with use of the LOG_FILE item) the contents of telegrams transmitted during data exchange between **asix** and Modbus Plus network controllers; writing the telegrams content to the log file should be used only during the **asix** system start-up.
Default value	- by default, the contents of telegrams isn't written to the log file.

☑ ***SEND_TIMEOUT=number***

Meaning	- defines timeout for sending a query to a controller; the timeout is given as a multiple of 0.5 second.
Default value	- by default, the item is set to 10 (5 seconds).

☑ ***RECV_TIMEOUT=number***

Meaning	- defines timeout for receiving an answer from a controller; the timeout is given as a multiple of 0.5 second.
Default value	- by default, the item is set to 10 (5 seconds).

1.5. AS511 - Driver of AS511 Protocol for SIMATIC S5 PLCs

Driver Use

The AS511 driver is used for data exchange with SIMATIC S5 PLCs by means of a programmer interface. The transmission is performed with use of serial interfaces of standard serial ports of **asix** system computers provided with the RS232C converter – current loop 20 mA. The operation of **asix** with PLCs with use of the AS511 protocol doesn't require any controller's program adaptation.

The AS511 driver of the **asix** system may be used for data exchange with the following PLC types: S5-90U, S5-95U, S5-100U, S5-115U, S5-135U.

Declaration of Transmission Channel

The syntax of declaration of transmission channel operating according to the AS511 protocol is given below:

logical_name=AS511,*port,[baud,character,parity,stop]*

where:

port	- serial port name;
baud	- transmission speeds in bauds; the transmission speed must be equal to 9600 bauds;
character	- number of bits in a transmitted character;
parity	- parity check type (even, odd, none).

The parameters *baud*, *character*, *parity*, *stop* i *buffer* are optional. When they are omit, the default values are as follows:

- transmission speed - 9600 Bd,
- number of bits in a character - 8,
- parity check type - parity check,
- number of stop bits - 2.

EXAMPLE

An exemplary item defining the use of transmission channel operating according to the AS511 protocol is given below:

```
CHAN1=AS511,COM1
```

The transmission channel with the logical name CHAN1 has the following parameters defined:

- AS511 protocol using a serial interface,

- port COM1,
- transmission speed of 9600 Bd,
- transmitted character length - 8 bits,
- parity check,
- two stop bits.

Addressing the Process Variables

The syntax of symbolic address which is used for variables belonging to the AS511 driver channel is as follows:

variable_type [db_number.]variable_index

where:

variabletype	- string identifying the variable type in the controller;
db_ number	- optional number of a data block; it is used only in case of process variables which map the content of words in data blocks;
variable_index	- variable index within a given type. In case of data blocks it is the word no. in a data block.

The following symbols of process variables are allowed:

EA	- states of outputs, transferred in bytes,
EAW	- states of outputs, transferred in words,
EE	- states of inputs, transferred in bytes,
EEW	- states of inputs, transferred in words,
EM	- states of marks (flags), transferred in bytes,
EMW	- states of marks (flags), transferred in words,
EZ	- states of counters, transferred in words,
ET	- states of clocks, transferred in words,
ED	- values of words in data blocks,
EL	- values of double words in data blocks,
EG	- values of double words in data blocks, treated as a floating-point number in KG format,

EXAMPLES

ED10.22	- word no. 22 in the data block no. 10
EL20.32	- double word placed in words no. 32 and no. 33 in the data block no. 20
EZ50	- counter no. 100

Driver Configuration

☑ ***BLOCK= YES/NO***

Meaning	***-*** *allows reading the whole data block.*
Default value	*- YES*

NOTE *The AS511 driver (from 1.23 version) allows reading words of data placed in data blocks by reading the whole block instead of determined part of cache (like it was in the previous version). It allows data reading from the 115F controller. Reading the whole block is possible if the parameter 'block' is placed in the INI file in the [AS511] section.*

1.6. AS512 - Driver of AS512 Protocol for SIMATIC S5 PLCs

Driver Use

The AS512 driver is used for data exchange with SIMATIC S5 PLCs provided with the CP524/CP525 communication processor. The transmission is performed with use of serial interfaces of standard or additional serial ports of a computer.

The controller software must be prepared for cooperation with **asix**, i.e.

- program in a CPU controller must include calls of functional blocks handling receiving and sending telegrams with use of a CP524/CP525 communication processor (*SEND_ALL*, *RECV_ALL*). The number of calls of these blocks within a controller operation cycle define the number of telegrams, which may be sent during the cycle between the computer and controllers !
- software of communication processor must use the 3964R procedure and transmission speed must be the same as the rate declared in the transmission channel of ASMEN.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel operating according to the AS512 protocol is given below:

logical_name=AS512,port, [,baud,character,parity,stop,cpu]

where:

port	- serial port name,
baud	- transmission speed in bauds,
character	- number of bits in a transmitted character,
parity	- parity check type (even,odd,none),
stop	- number of stop bits,
cpu	- CPU number in the controller, to which the carried out operation refers.

The parameters *baud*, *character*, *parity*, *stop*, *cpu* i *buffer* are optional. In case of omitting them the default values are as follows:

- transmission speed - 9600 Bd,
- number of bits in a character - 8,
- parity check type - parity check,
- number of stop bits - 1,
- CPU number - 0.

EXAMPLE

An example item, which defines the use of transmission channel operating according to the AS512 protocol, is given below:

```
CHAN1=AS512,COM1,4800,8,even,1,2
```

The transmission channel of the logical name CHAN1 has the following parameters:

- AS512 protocol using a serial interface,
- port COM1,
- transmission speed of 4800 Bd,
- transmitted character length - 8 bits,
- parity check,
- one stop bit,
- data exchange concerns the CPU no. 2.

Addressing the Process Variables

During declaration of process variable its symbolic address is entered. It is used as an unique definition of the controller variable, the value of which will be assigned to the process variable in **asix**. The syntax of symbolic address which is used for the variables belonging to the AS512 driver channel, is presented below:

variabke_type [db_number.]variable_index

where:

variable_type	- string identifying the variable type in the controller;
db_number	- optional number of a data block; it is used only in case of process variables which are the content mapping of words in data blocks;
variable_index	- variable index within a given type. In case of data blocks, it is the word no. in a data block.

The following symbols of process variables types (following the names of variable types used by SIEMENS) are permitted:

EA	- states of outputs, transferred in bytes,
EE	- states of inputs, transferred in bytes,
EM	- states of marks (flags), transferred in bytes,
EZ	- states of counters, transferred in words,
ET	- states of clocks, transferred in words,
ED	- values of words in data blocks,
EL	- values of double words in data blocks,
EG	- values of double words in data blocks, treated as a number in the KG floating-point format.

EXAMPLES

ED10.22	- word no. 22 in the data block no. 10
EZ100	- counter no. 100

The AS512 driver is loaded as a DLL automatically.

1.7. AS512S7 - Driver of AS512 Protocol for SIMATIC S7 PLCs

Driver Use

The AS512S7 driver is used for data exchange with SIMATIC S7 PLCs provided with the CP340 communication processor. The transmission is performed with use of serial interfaces in standard serial ports of a computer according to the AS512 protocol.

The ASKOM company offers the software for the SIMATIC S7 PLC that enables data exchange with **asix** according to the AS512 protocol.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel operating according to the AS512S7 protocol is given below:

logical_name=AS512S7,*port, [,bauds,character,parity,stop,cpu]*

where:

port	- name of the serial port,
bauds	- transmission speed in bauds,
character	- number of bits in a transmitted character,
parity	- parity check type (even, odd, none),
stop	- number of stop bits,
cpu	- number of the CPU (to which the carried out operation refers) in the controller.

The parameters *bauds*, *character*, *parity*, *stop*, *cpu* are optional. When they are omitted, the default values are as follows:

- transmission speed - 9600 Bd,
- number of bits in a character - 8,
- parity check type - parity check,
- number of stop bits - 1,
- CPU number - 0.

EXAMPLE

An example item, which defines the use of transmission channel operating according to the AS512S7 protocol, is given below:

CHAN1=AS512S7,COM1,4800,8,even,1,2

The transmission channel of the logical name CHAN1 has the following parameters defined:

- AS512S7 protocol using a serial interface,

- port COM1,
- transmission speed of 4800 Bd,
- transmitted character length - 8 bits,
- parity check,
- one stop bit,
- data exchange concerns the CPU no. 2.

During the declaration of process variable its symbolic address is entered. It is an unique definition of the controller variable, the value of which will be assigned to the process variable in **asix**.

Addressing the Process Variables

The syntax of symbolic address which is used for the variables belonging to the AS512S7 driver channel is presented below:

variabke_type [db_number.]variable_index

where:

variable_type	- string identifying the variable type in the controller;
db_number	- optional number of a data block; it is used only in case of process variables, which are the content mapping of words in data blocks;
variable_index	- variable index within a given type. In case of data blocks, it is the word no. in a data block.

In the AS512S7 protocol only the access to words in data blocks is implemented. For this reason there is only one type of process variables allowed:

ED	- values of words in data blocks.

EXAMPLES

ED10.22	- word no. 22 in the data block no. 10

The AS512S7 driver is loaded as a DLL automatically.

1.8. BAZA - Driver for Access to Database

Driver Use

The BAZA driver allows to import data into the **asix** system from databases. The access to database is realized on the basis of the ADO technology. BAZA makes all the data stored in databases available to the **asix** system. The received data may (but they needn't) be stamped with a status and time. The driver also allows reading from other sources like an Excel spreadsheet. If data are stamped with a time, the driver allows to complete historical data in ASPAD archives. When data stored in a database are not stamped with a time, the data newly received by the driver are stamped with a current time. If a datum is not stamped with a status, the status *proper datum* will be assigned to it.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel operating with the BAZA driver:

logical_name=BAZA, *database*

where:

database	- field that determines a database. It can be: - file name; the name has to allow to distinguish it from the database name i.e. it has to include the "." or "\" character; when declaring a file name as *database*, it is assumed that it is a Microsoft Jet database (Microsoft.Jet.OLEDB.4.0); - database name; in such case the database is assumed to be serviced by an SQL Server on a local station (SQLOLEDB); - connection string put in quotation marks; this specification form allows to determine an arbitrary database, i.e. to determine the following parameters: database server localization (e.g. remote computer), user name, password, timeout of establishing a connection etc.; this form allows to specify a database as a DSN name; - section name, put in square brackets, in which elements of the connection string are placed; this form is used in case of a long connection string.

EXAMPLE

An exemplary declaration of the transmission channel:

```
[ASMEN]
....
;Microsoft Jet database:
Measurements1 = BAZA, c:\Pomiary.mdb

;Database defined by DSN data source (of computer or user)
Measurements2 = "DSN=Pomiary"

;Database defined by file DSN data source
Measurements3 = "FILEDSN=C:\BAZA\Pomiary.dsn"

;Database defined by UDL file (Microsoft Data Link)
Measurements4 = Baza,"File Name=C:\BAZA\Pomiary.UDL"

;SQL database named „Pomiary" on local computer:
Measurements5 = BAZA, Pomiary

;SQL database named "Pomiary" on computer "Emisja"
Measurements6 = BAZA,"Provider=SQLOLEDB.1;Data Source=Emisja;Initial
Catalog = Poiary;Integrated Security=SSPI;"

;Parameters of SQL database named „Pomiary" in separate section
Measurements7 = BAZA,[BAZA-POMIARY]

[BAZA-POMIARY]
Provider = SQLOLEDB.1
Data Source = Emisja
Initial Catalog = Pomiary
Integrated Security = SSPI
```

Addressing the Variables

The syntax of symbolic address which is used for the variables belonging to the BAZA driver channel is presented below:

array_declaration[,value_field[.[time_field][.status_field]]]

where:

array_declaration	- expression that determines an array in the database (set of records);
value_field	- field name (column) containing the datum value;
time_field	- field name (column) containing the data time (Data/Time type);
status_field	- field name (column) containing the data status (numerical type – OPC status).

Value_field may be omitted if *array_declaration* determines the array containing one column.

If *time_field* is omitted, a current time is taken.

If *status_field* is omitted, the *proper data* status is assigned to the variable value.

Array_declaration can be:

- name of the array placed in the database;
- query, put in apostrophes or round brackets, sending to the database by the driver in order to data readout;
- symbolic name in the $(*name*) form. The name determines the query the syntax of which is defined in the [BAZA] section in an initialization file.

In the most simple and typical case the array is determined by its name. If, for example, the database contains the array named Pomiary containing Temperature, Pressure, Time and Status columns, then variable addresses would have the following form:

```
Pomiary.Temperature.Time.Status
Pomiary.Pressure.Time.Status
```

EXAMPLE

When the more complex rule defining the records is necessary, other address forms may be used, i.e. with query text, e.g.:

```
(SELECT * FROM Pomiary WHERE ......). Temperature.Time.Status
```

The queries have to be constructed to determine a record set ordered decreasingly according to the time. When reading current data, the driver modifies the query to read the newest record (the driver adds TOP 1 fraze). When reading historical data (only when address contains *time* field), the driver adds or modifies the WHERE fraze to receive data from the specified time range.

The other form of using the queries is use of a query name. The query is defined in the [BAZA] section in an initialization file. The name use allows:

- shortening the address in case of using many variables with the same query but different value fields;
- avoidance of errors in case of necessity of using the characters (which are interpreted by ASMEN in a different manner) in the query;
- optimization of a query number, i.e. if the array of many variables is the result of the query, then the driver sends only one query instead of one by one for each variable.

An exemplary declaration of query name use:

Definition of the variable address:
Name: Temperature
Address: $(QUERY1).Temperature.Time.Status

Name: Pressure
Address: $(QUERY1).Pressure.Time.Status

Initialization file:

```
[BAZA]
Query1 = SELECT * FROM Pomiary WHERE .....
```

NOTICE *The quotation marks (") should NOT be used in the address.*

Driver Configuration

The BAZA driver may be configured using the **[BAZA]** section placed in the application initialization file or sections having the same name as channel names in the definition of channels in the [ASMEN] section. The parameters placed in the [BAZA] section apply to all driver channels. The parameters placed in other sections refer to a specified channel. If the parameter is defined both in the [BAZA] section and in the channel section, then the parameter referring to a specified channel has a higher priority.

☑ ***No_TOP =YES/NO***

Meaning	- if the item has value NO, then the driver puts in the SQL query the fraze TOP limiting a number of read records. Some databases don't allow using the fraze TOP – then one should declare the value YES.
Default value	- NO.
Log =log_file	
Meaning	- defines a file to which all the diagnostic messages of the driver and the information about the content of telegrams received by the driver will be written.
Default value	- lack.

☑ ***Max_history=number***

Meaning	- determines a time period from the current moment backwards, for which historical data, saved in the station memory, will be read.
Default value	- 30.
Parameter:	
number	- number in days.

☑ ***Record_optimize=number***

Meaning	- the parameter refers to variables determined by the name of the array. If the item has the value Yes, then only one (common for all variables) SQL query, causing readout of the record containing only the fields that occur in variable addresses, will be formulated for all variables placed in the array. If the parameter has the value No, all fields in array will be read.
Default value	- yes.

☑ ***Order =yes/no***

Meaning	- if the parameter has the value Yes, the driver will formulate an SQL query in such way that read records will be ordered in according to the time fields. If the parameters has the value No, the records will not be ordered.
Default value	- yes.

☑ ***History_records =number***

Meaning	- the parameter determines the maximal number of records read from the database once during historical data reading. The parameter has a sense only when *No_TOP* parameter takes the value No.
Default value	- 1000.
Parameters:	
number	- number of records.

☑ ***UTC =yes/no***

Meaning	- determines whether the time written in the database is an UTC time (Universal Time Coordinate or Greenwich Mean Time). If the parameter has the value Yes, the time is an UTC time, if the parameter has the value No – the time is a local time.
Default value	- no.

Optimization of Field Number in a Record

The driver formulates one SQL query, common for all variables contained in the same array, causing readout of record including only the fields that occur in variable addresses. If the address even of one variable includes the name of the field that doesn't occur in the array, then reading of all array variables ends with an error. To determine which of variables has an incorrect name, one should declare *Record_optimize=No* - thanks to such solution the error will refer only to wrongly declared variables. If the error reffers to the time field name, one should additionally set the *Order=No* parameter - but absence of ordering may cause reading incorrect data.

1.9. BUFOR – Driver of General Purpose

Driver Use

The BUFOR driver allows data exchange between the **asix** system and any program developed by the user to transfer the process variables.

The transmission channel based on the BUFOR protocol is created by two programs:

- driver of the protocol BUFOR for general purpose (included in the **asix** package),
- driver of process data transmission developed by the user.

The user program must be implemented in the form of a process operating in Windows XP / 2000 / NT 4.0 environments.

The data exchange between the BUFOR driver and the user program is performed by means of an exchange file (memory mapped file). The synchronization of the access to a memory mapped file is carried out by using a mutex object.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel operating with the BUFOR driver is given below:

logical name =BUFOR,FILE_MMF,USER,PAR1,PAR2,PAR3

where:

FILE_MMF	- memory mapped file name;
USER	- user process name;
PAR1..PAR3	- parameters which are transferred to the user process.

The description of data structures and rules of cooperation of a user process with the BUFOR driver is included in the file DrBufor.hlp.

Addressing the Process Variables

The syntax of symbolic address which is used for process variables belonging to the BUFOR driver channel is as follows:

Ivariable_index

where:

I	- constant element of symbolic address for the BUFOR driver channel;

variable_index - variable index in an array of user driver variables. The first variable has the index 1.

The other parameters in a process variable declaration have typical meaning.

The BUFOR driver is loaded as a DLL automatically.

Possibility of Mutex Name Declaration

It is possible to declare a mutex name in the MUTEX item according to the following syntax (a mutex object synchronizes the driver BUFOR with a user's program at the stage of start-up):

MUTEX=*mutex_name*

It is necessary to put the position in a separate section of the initialization file of an **asix** application. The section name has to be identical to the name of logical channel using the driver BUFOR, e.g.:

[ASMEN]
# item declaring the transmission channel CHANNEL
CHANNEL=BUFOR, *TEST_MMF*, *userebuf*, *par1 par2 par3*

# this section is used by designer to the parameterization of CHANNEL
# (in the present version of **asix** it is used for an optional mutex name declaration only)
[CHANNEL]
MUTEX=TEST_MUTEX

When a mutex name is declared, the BUFOR driver operates according to the scheme:

- the driver creates the user's process by means of the procedure *CreateProcess()* and expects that the process will create a mutex with the name given in the MUTEX position; while creating a mutex, the process must occupy it at once!

- while giving the mutex name declared in the MUTEX position, the driver calls the *OpenMutex()* function; if *OpenMutex ()* returns an error, the driver will finish the channel initialization with the error;

- the driver expects till the user's program releases mutex and after that realizes the stage of verification of memory mapped file contents.

1.10. CTCalec - Driver of CALECMCP Device Protocol

Driver Use

The CTCalec driver is designed to retrieve current values of variables from the CALEC MCP meter of Aquametro.

CTCalec is a dynamic link library (DLL) with an interface meeting the requirements of the UniDriver module. The CTCalec driver together with UniDriver create a driver meeting the requirements of the ASMEN module.

Declaration of Transmission Channel

The ASMEN module activates the driver after having found (in the ASMEN section of the **asix** application configuration file) the channel declaration, calling to CTCalec, of the following form:

channel_name = UniDriver, CTCalec, *driver parameters*

where:

channel_name - channel name of ASMEN;
driver parameters - configuration parameters of CTCalec described in the further part of the specification.

During loading and initialization of an **asix** application, CTCalec receives from the ASMEN module an address of each process variable retrieved from the source definition of variables (in sequence and one time). The variables supplied by the driver are read-only.

<u>Configuration Parameters</u>

The driver supports the following configuration parameters described in the following table.

Table 6. Configuration Parameters of CTCalec.

Parameter Name	Meaning	Default Value
Port	Indicates the name and parameters of the serial port the device is connected to. For detail description, see below.	N/A– parameter required
ReadingPeriod	Indicates the period (in seconds); a variable value will be updated every *ReadingPeriod* in the internal driver buffer.	10 s
ReadingTimeout	The maximal time of variable reading from the device (in milliseconds), it is recommended not to be less than 1200.	1200 ms

EXAMPLE

An example of the CTCalec channel declaration (all items must be written in one line):

CalecMCP = UniDriver, CTCalec, Port=COM1:9600:8:even:1, ReadingPeriod=10, ReadingTimeout=1200

Configuration Parameters of Serial Port

The full syntax of the serial port declaration is as follows:

Port=<*port name* >:<*rate*>:<*character*>:<*parity*>:<*stop*>

where:

port name	- serial port name in the operation system, e.g. COM1 or COM2;
rate	- baud rate of serial transmission;
character	- number of bits in a transmitted character;
parity	- parity check type (odd, even or none);
stop	- number of stop bits, it may be entered 1, 15 (1.5 bits) or 2.

The simplified syntax includes *port name* only:

Port = <*port name*>

For the other parameters default values are assigned respectively:
9600:8:even:1

Addressing the Process Variables

A variable address consists of a hexadecimal address of the variable in the device and of the type of variable value coding separated by a colon. As the type of variable value it may be assumed FLOAT for floating-point variables and FIX for fixed-point variables.

EXAMPLE

2000:FIX	Fixed-point variable to be retrieved from the address 2000 in hexadecimal format,
2080:FLOAT	Floating-point variable to be retrieved from the address 2080 in hexadecimal format.

For floating-point variables the conversion function NOTHING_FP should be used and for fixed-point variables the conversion function NOTHING.

Period of Variable Refreshing (Updating)

The reading time of one variable is usually in the range from 520 ms to 630 ms (average 570 ms) but for some variables (in a tested device it was the measurement named Temperaturdifferenz) it may reach 1300 ms. When defining a driver reading period and variable refreshing period it is necessary to take into consideration that it should be physically possible to read the required variable in the defined refreshing period. Generally the driver reading period and the variable refreshing period are the same.

Protocol Functions Excluded from Implementation

In the documentation it is mentioned that there is at least one *text* type variable in the device. Because the transmission method of this data type is not described, it is not supported by the driver.

1.11. CAN_AC_PCI - Driver of CANBUS Protocol for CAN ACx PCI Card

Driver Use

The CAN_AC_PCI driver is used for data exchange between SELECONTROL MAS PLCs of Selectron Lyss AG and **asix** system computers by using the CAN network. The **asix** system computer must be provided with a card of communication processor CAN_AC1 or CAN_AC2 of Softing GmbH.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel operating according to the CAN_AC_PCI driver is given below:

logical_name=CAN_AC_PCI,*interface_no*

where:

interface_no	- number identifying the CAN_AC1/CAN_AC2 card interface by means of which the transmission with the CAN network is executed. In the CAN_AC1 card the interface no. 1 may be used exclusively.

The CAN_AC_PCI driver is loaded as a DLL automatically.

Addressing the Process Variables

Values of process variables are transferred in telegrams sent by CAN based controllers connected to a CAN network. Each telegram consists of at most 8 bytes that may be identified as:

- bytes indexed 1 – 8	(type BY),
- 16-bit numbers indexed 1 – 4	(type WD),
- 32-bit numbers indexed 1 – 2	(type (DW).

The CAN_AC_PCI driver distinguishes the following types of access to process variables:

- read-only (type R_),

- write-only	(type W_),
- read/write	(type RW_).

The addressing of process variables consists in indication of:
- access type (R_, W_ or RW_);
- variable type (BY, WD, DW);
- telegram no. (for variables with RW_ access type it is the telegram number used to read the variable),
- index within the telegram (for variables with RW_ access type it is the index in the telegram used to read the variable),
- for variables with RW_ access type it is necessary to declare in addition:
  a/ telegram no. used to read the variable,
  b/ index in the telegram used to write variable.

The syntax of symbolic address which is used for variables belonging to the CAN_AC_PCI driver channel is as follows:

<access_type><variable_type><tel>.<index>[.<tel>.<index>]

where:

access type	- access type to the process variable:
R_	- read-only,
W_	- write-only,
RW_	- read/write,
variable_type	- process variable type:
BY	- variable of the byte type,
WB	- variable of the 16-bit number type,
DW	- variable of the 32-bit number type,
tel	- telegram no.,
index	- index within the telegram.

EXAMPLE

```
X1, byte no 2 of telegram 31,R_BY31.2,          NONE, 1, 1, NOTHING_BYTE
X2, word no 3 of telegram 31,R_WD31.3,          NONE, 1, 1, NOTHING
X3, burner state,       RW_BY31.1.35.3,         NONE, 1, 1, NOTHING_BYTE
X4, valve setting,RW_WD32.1.34.1,               NONE, 1, 1, NOTING
```

Value of the variable X3 is transferred to the **asix** system by means of the byte no. 1 of the telegram no. 31. The transfer of the variable X3 consists in sending from **asix** a telegram no. 35, the byte no. 3 of which includes the required state of the variable X3.

Driver Configuration

The CAN_AC_PCI protocol driver may be configured by using the **[CAN_AC_PCI]** section of the application INI file. Individual parameters are given in separate items of the section. Each item has the following syntax:

item_name=[number [,number]] [YES] [NO]

☑ ***TRANSMISSION_SPEED=interface_no,baud_id***

Meaning	- the item is used to declare a transmission speed in the CAN network.

Default value	- by default, the transmission speed is assumed to be 1 MB.
Parameter:	
interface_no	- interface no. of the CAN_AC card (for the CAN_AC1 card always 1),
baud_id	- identifier of transmission speed in the CAN network:

1	- 1 MB
2	- 500 kB
3	- 250 kB
4	- 125 kB
5	- 100 kB
6	- 50 kB
7	- 20 kB

EXAMPLE

An example of declaration of transmission speed of 20 kB (the CAN network numbered 1):

```
TRANSMISSION_SPEED=1,7
```

☑ ***REFRESH_CYCLE=number***

Meaning	- the item is used to declare a time interval between two successive signals allowing the CAN_AC card driver to read data from the CAN network.
Default value	- by default, the CAN_AC_PCI driver reads data every 0.5 second.
Parameter:	
number	- number of 0.5-second intervals, which must pass between two successive signals allowing the CAN_AC board driver to read data from the CAN network.

EXAMPLE

An example of declaration of data reading every 1 second:

```
REFRESH_CYCLE=2
```

☑ ***NETWORK_CONTROL=number***

Meaning	- the item allows to test the reception of telegrams from the CAN network. It defines the maximal time (in seconds) between reception of successive telegrams with the same number. In case of exceeding this time the process variables from such telegram will be provided with an error status. If additionally in the same time any telegram wasn't received from the CAN network the message about a lack of telegrams in the network is generated in *'Control Panel'*.
Default value	- by default, the CAN_AC_PCI driver doesn't check reception of telegrams.
Parameter:	
number	- maximal number of seconds, which may pass between successive telegrams of the same number.

EXAMPLE

An example of checking the reception of telegrams every 5 seconds:

```
NETWORK_CONTROL=5
```

☑ ***TELEGRAM_TRACE=YES|NO***

Meaning	- the item controls transferring to the operator panel the messages about telegrams that have been received from the CAN network. A message includes the number of CAN network, the number of telegram, number of bytes and telegram contents in hexadecimal form.
Default value	- by default, the contents of telegrams are not displayed.

☑ ***CONTROL_TRACE=YES|NO***

Meaning	- the item controls transferring to the operator panel the messages about control telegrams that have been sent from the **asix** system computer to controllers. A message includes the number of CAN network, the number of telegram, number of bytes and telegram contents in hexadecimal form.
Default value	- by default, the contents of telegrams are not displayed.

☑ ***LOG_FILE=file_name***

Meaning	- the item allows to define a file to which all messages, describing the telegrams received from the CAN network, will be written. If LOG_FILE does not define the full path, then the log file will be created in the current directory. The log file should be used only while the **asix** start-up.
Default value	- by default, the log file is not created.

☑ ***USE_CONTROL_VALUE=YES|NO***

Meaning	- the driver has two pools of telegrams: sending and receiving ones. The sending telegrams are used by **asix** to send control data whereas the receiving telegrams contain actual copies of telegrams sent from controllers and are a source of values of process variables for **asix**. USE_CONTROL_VALUE allows to copy a value of the process variable (of W type) from a sending telegram directly to a buffer of the receiving telegram. The copying concerns the receiving telegram with the same number as the sending telegram assigned to the control variable and is executed only after that have done the control operation correctly. In this way the values of process variables are used by driver as actual values of process variables. This state lasts up to the moment the real values of process variables under consideration will be read from a controller.

	The CONTROL_VAR_CHECK item allows to change the copying mode of control values.
Default value	- by default, the control values are not copied to the buffers of receiving telegrams of the driver.

Checking Control Variables

The declaration of W_ type variables (control variables) are checked by default. It is possible to use only one variable of this type in one telegram.

☑ ***CONTROL_VAR_CHECK =YES|NO***

Meaning	- the item permits to change default settings and enables using telegram to send control value by means of more than one variable of W_ type. Individual control values are send sequentially, i.e. by sending a separate telegram transferring a value of one control variable only, the other parts of the telegram are filled in with zeroes.

Change of Default Settings for Updating Mode of Receiving Buffers by Means of Control Variables

☑ ***USE_RW_DECLARATION=YES|NO***

Meaning	- the item allows to change the mode of copying control variables to buffers of driver receiving telegrams. The item is import for W_ type variables only, and is effective if it is used together with the CONTROL_VAR_CHECK=YES item. The result of using the item under consideration is copying the control variable to the buffer of receiving telegram (with the number specified in the process variable declaration in the position of telegram that is used to read the variable value).
Default value	- by default, the driver buffers are not updated according to declarations of RW_ type variables.

1.12. CANOPEN – Driver of CANBUS Protocol for PCI 712 NT Card

Driver Use

The CANOPEN driver is used for data exchange between SELECONTROL MAS PLCs of Selectron Lyss AG and **asix** system computers by using the CAN network. The **asix** system computers must be provided with the PCI_712 NT communication processor board and the PCI712 CanLib32 software package of Selectron Lyss AG.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel working according to the CANOPEN protocol is given below:

logical_name=CANOPEN,*card_no*

where:

card_no	- PCI_712 NT card no., by means of which the transmission with the CAN network is executed. In the present version the CANOPEN driver can operate with one PCI_712 NT board.

The CANOPEN driver is loaded as a DLL automatically.

Addressing the Process Variable

Values of process variables are transferred in telegrams sent by controllers connected to the CAN network. Each telegram consists of at most 8 bytes, which can be identified as:

bits with indexes 1 – 8	(type BY),
16-bit numbers with indexes 1 – 4	(type WD),
32-bit numbers with indexes 1 – 2	(type (DW).

The CANOPEN driver distinguishes the following types of access to process variables:

read-only	(type R_),
write-only	(type W_),
read/write	(type RW_).

Addressing the process variables consists in indication of:

- access type (R_, W_ or RW_);
- variable type (BY, WD, DW);
- telegram no. (for the variables with RW_ access type it is the number of telegram used to read the variable);
- index within the telegram (for the variables with RW_ access type it is the index of telegram used to read the variable);
- for the variables of RW_ access type it is necessary to declare additionally:
    - number of telegram used to write the variable,
    - index of telegram used to write the variable.

The syntax of symbolic address which is used for variables belonging to the CANOPEN driver channel is as follows:

<access_type><variable_type><tel>.<index>[.<tel>.<index>]

where:

access_type	- type of access to process variables:
R_	- read-only,
W_	- write-only,
RW_	- read/write,
variable_type	- type of process variables:
BY	- variable of byte type,
WB	- variable of 16-bit number type,
DW	- variable of 32-bit number type,
Tel	- telegram no.,
Index	- index within the telegram.

EXAMPLE

```
X1, byte no 2 of telegram 31,  R_BY31.2,     NONE, 1, 1, NOTHING_BYTE
X2, word no. 3 of telegram 31,R_WD31.3,     NONE, 1, 1, NOTHING
X3, state of burners,RW_BY31.1.35.3,        NONE, 1, 1, NOTHING_BYTE
X4, valve setting,RW_WD32.1.34.1,           NONE, 1, 1, NOTHING
```

The X3 variable value is transferred to **asix** in byte no. 3 of the telegram 31. The setting of X3 variable consists in sending from **asix** the telegram no. 34, the byte no. 3 of which contains required value of X3 variable.

Driver Configuration

The driver of the CANOPEN protocol may be configured with use of the **[CANOPEN]** section in the initialization file of an **asix** application. Individual parameters are transferred in separate items of the section. Each section has the following syntax:

item_name=[number [,number]] [YES|NO]

☑ ***TRANSMISSION_SPEED=network_no,baud_id***

Meaning	- the item is used to declare the transmission speed in the CAN network.

Default value	- by default, the transmission speed is assumed to be 1 MB.
Parameter:	
network_no	- CAN network no. (in the present version always the network no. 1),
baud_id	- identifier of transmission speed in the CAN network:

1	- 1 MB
2	- 500 kB
3	- 250 kB
4	- 125 kB
5	- 100 kB
6	- 50 kB
7	- 20 kB

EXAMPLE

An example of declaration of transmission speed of 20 kB (the CAN network numbered 1):

```
TRANSMISSION_SPEED=1,7
```

Frequency of Data Reading from PCI_712 NT Card

☑ ***REFRESH_CYCLE=number***

Meaning	- the item is used to declare a time interval between successive signals allowing the PCI_712 NT card driver to generate information on messages received from the CAN network.
Default value	- by default, the CANOPEN driver send signals every 0.5 sec.
number	- equal to 0.5-second intervals, which must pass between successive signals allowing the PCI_712 NT card driver to generate information on messages received from the CAN network.

EXAMPLE

The declaration of sending a permission signal every 1 sec:

```
REFRESH_CYCLE=2
```

Checking Reception of Telegrams from CAN Network CAN

☑ ***NETWORK_CONTROL=number***

Meaning	- the item allows to test reception of telegrams from the CAN network. It defines the maximal time (in seconds) between receptions of successive telegrams with the same number. In case of exceeding this time the process variables bound with such telegram will be provided with an error status. If additionally in the same time any telegram wasn't received from the CAN network a message about a lack of telegrams in network is generated in *'Control Panel'*.

Default value	- by default, the CANOPEN driver doesn't check reception of telegrams.
number	- maximal number of seconds which may pass between successive telegrams with the same number.

EXAMPLE

An example of checking reception of telegrams every 5 seconds:

```
NETWORK_CONTROL=5
```

Tracing Telegrams Received from CAN Network

☑ ***TELEGRAM_TRACE=YES|NO***

Meaning	- the item controls transferring to the operator panel the messages about telegrams that have been received from the CAN network. A message includes the number of CAN network, the number of telegram, number of bytes and telegram content in hexadecimal form.
Default value	- by default, the contents of telegrams are not displayed.

EXAMPLE

The declaration of tracing the received telegrams:

```
TELEGRAM_TRACE=YES
```

Tracing of Control Telegrams

☑ ***CONTROL_TRACE=YES|NO***

Meaning	- the item controls transferring to the operator panel the messages about control telegrams that have been sent from the **asix** system computer to controllers. A message includes the number of CAN network, the number of telegram, number of bytes and telegram contents in hexadecimal form.
Default value	- by default, the contents of telegrams are not displayed.

EXAMPLE

The declaration of tracing the control telegrams:

```
CONTROL_TRACE=YES
```

☑ ***LOG_FILE=file_name***

Meaning	- the item allows to define a file to which all the messages, describing telegrams received from the CAN network, will be written. If LOG_FILE does not define the

	full path, a log file will be created in the current directory. The log file should be used only while the **asix** system start-up.
Default value	- by default, the log file is not created.

EXAMPLE

```
LOG_FILE=D:\asix\CAN.LOG
```

1.13. COMLI - Driver of COMLI Protocol

Driver Use

The driver is designed for data exchange between the **asix** system and SattCon*xx* and other ABB PLCs supporting the COMLI (**COM**munication **LI**nk) protocol. The data exchange is performed by means of a serial interface in the RS-232 or RS-485 standard.

In the present driver version the following functions of COMLI protocol are implemented:

- Transfer individual I/O bits,
- Transfer I/O bits or a register,
- Request individual I/O bits,
- Request several I/O bits or registers,
- Transfer date and time,
- Acknowledge.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel working according to the COMLI protocol is given below:

channel_name=COMLI, *slave_no, port [, baud] [,parity] [,data_type]*

where:

COMLI	- driver name;
slave_no	- slave no. assigned to the PLC;
port	- name of the serial port through which the connection with the PLC will be executed;
baud	- option: transmission speed; by default 9600;
parity	- option: parity check; by default ODD;
data_type	- option: representation of ASCII data (ASCII) or binary (BINARY); by default it is BINARY.

Below there is an example of declaration of transmission channel named CHANNEL, that is used for communication with the controller no. 3 through the port COM2 in default mode, i.e. 9600 Bd, 8 bit character, check parity ODD, binary representation of data:

```
CHANNEL = COMLI, 3, COM2
```

Types of Process Variables

In the driver the following types of process variables are defined:

IO	- I/O states,
RG	- values of registers,
TM	- writing date and time to a PLC.

The values of **IO** and **RG** type variables may be read and written whereas the values of **TM** type variable may only be written.

Addressing the Process Variables

The syntax of symbolic address which is used for variables belonging to the COMLI driver channel is as follows:

<Type><Index>

where:

Type	- name of variable type,
Index	- address of the variable within the *Type* type of the variable.

Ranges of indexes are following:

type **IO**:	0 – 16383,
type **REG**:	0 – 3071,
type **TM**:	no index is given.

EXAMPLE

Examples of declaration of process variables:
JJ_1, state I/O number 1,IO1,CHANNEL,1,1,NOTHING
JJ_2, register number 10,RG10,CHANNEL,1,1,NOTHING
JJ_40, writing date and time every 1 minute, TM,
CHANNEL,12,60,NOTHING_BYTE

Synchronization of Date and Time with the Controller

In the driver a mechanism of synchronization of date and time between **asix** and PLCs is included. The synchronization is activated for each transmission channel separately with use of items included in the ASMEN section:

TIME_SYNCHRONIZATION = *channel, variable*

where:

channel	- name of transmission channel used for communication with defined PLC;
variable	- name of an ASMEN variable belonging to the channel *channel* and using for synchronization of date and time.

The synchronization of date and time consists in writing to a PLC a frame containing the current date and time of **asix**. The frame is written by using the function for writing the date and time of the COMLI protocol according to the frequency assigned to the *variable*. The variable type must be the type **TM** (supporting date and time), the number of elements assigned to the *variable* must be 12 (the size of frame of date and time). The function NOTHING_BYTE must be used as a calculation function.

EXAMPLE

An example of the definition of time synchronization every 1 minute for the channel *CHANNEL* by using the variable SYNCHRO is given below.

```
[ASMEN]
DATA= COMLI.DAT
CHANNEL = COMLI, 2, COM1
TIME_SYNCHRONIZATION = CHANNEL, SYNCHRO
```

The declaration of the variable SYNCHRO is found in the file *COMLI.DAT* and has the following form:

```
SYNCHRO, synchronization of date and time, TM, CHANNEL,12, 60,
NOTHING_BYTE
```

Driver Configuration

The driver configuration is performed by using a separate section named **[COMLI]**. By means of this section it is possible to declare:

- wait timeout for a response from a PLC,
- log file and its size,
- log of telegrams,
- number of repetitions.

☑ ***RECV_TIMEOUT=slave_no, number***

Meaning	- for each slave it is possible to define the maximal time, which may elapse between sending query and receiving response (so-called receiving timeout).
Default value	- by default, the item has a value of 2000.
Parameter:	
slave_no	- slave no. assigned to the controller.
number	- value of timeout in milliseconds.

☑ ***LOG_FILE=file_name***

Meaning	- the item allows to define a file to which all the diagnostic messages of the driver and the information about the content of telegrams sent/received by the driver will be written. If the item doesn't define a full path the log file will be created in the current directory. The log file should be used only while the **asix** start-up.
Default value	- by default, the log file is not created.

☑ ***LOG_FILE_SIZE=number***

Meaning	- the item allows to define the size of the log file in MB.
Default value	- the default size of the log file amounts 1 MB.
Parameter:	
number	- size of the log file in MB

☑ ***LOG_OF_TELEGRAMS =YES|NO***

Meaning	- the item allows to write the content of telegrams sent/received by the driver to the log file (declared by means of the item LOG_FILE). Writing the content of telegrams should be used only while the **asix** start-up.
Default value	- by default, the driver does not write the content of telegrams to the log file.

☑ ***NUMBER_OF_REPETITIONS=number***

Meaning	- the item allows to define maximal number of trials to do the command in case of transmission errors.
Default value	- by default, max. 3 repetitions are executed.
Parameter:	
number	- number of repetitions.

1.14. DataPAF - Driver of DataPAF Energy Counter Protocol

Driver Use

The DataPAF protocol is designed for communication with DataPAF energy counters provided with the EPROM 2.1 13.Aug.96 or later one. For the communication a COM interface is used.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel working according to the DataPAF protocol is given below:

logical_name=DataPAF,*COMn*

where:

COMn	- number of the serial port to which the DataPAF energy counter is connected.

Each defined channel may have its own section, the name of which is a logical name of the channel. It contains parameters for the given channel. Some channels may have a common serial port.

Also the given port COMn may have its own section named [DataPAF:*n*]. It defines parameters of the serial port.

Driver Configuration

The default values of serial interfaces are retrieved from the section **[DataPAF]**, if section exists. This section is used to configure all the channels for the DataPAF protocol, which are declared in this system.

☑ ***NUMBER =number***

☑ ***serial_Number=number***

Meaning	- determines an identification number of the DataPAF counter. Input of incorrect number makes the communication impossible.
Default value	- 1234.
Parameter:	

number	- counter number.

☑ ***baud =number***

☑ ***bps=number***

Meaning	- determines transmission speed.
Default value	- 4800.
Parameter:	
number	- transmission speed in Bd.

☑ ***diagnostics =number***

Meaning	- declaring this position with value 14 will cause outputting the diagnostic information connected with time synchronization to the log file.
Default value	- 0.
Parameter:	
number	- 14.

☑ ***parity =parity_parameter***

Meaning	- determines parity;
Default value	- N.
Parameter:	
parity_parameter	- allowed value:

n	- no parity bit,
o	- odd parity check,
e	- even parity check,
m	- mark,
s	- space.

☑ ***stop_bits =number***

Meaning	- determines a number of stop bits.
Default value	- 1.
Parameter:	
number	- allowed values are 1 and 2.

☑ ***word =number***

☑ ***word_length=number***

Meaning	- determines word length.
Default value	- 8.
Parameter:	
number	- allowed values are from the range of 5 to 8.

time_out =number

timeout =number

Meaning	- waiting time for the DataPAF answer.
Default value	- 1000.
Parameter:	
number	- in miliseconds.

repetitions =number

Meaning	- number of repetitions of communication operations ended with an error.
Default value	- 3.
Parameter:	
number	- number of repetitions.

max_Time_Difference =number

Meaning	- determines the maximal difference between the **asix** system time and the DataPAF counter time, after which warnings will be generated in *‘Control Panel’*. The station time can be read with an interval defined by the parameter *time_Check*.
Default value	- 60.
Parameter:	
number	- number in seconds.

time_Check =number

Meaning	- interval, with which the current time of the counter is read.
Default value	- 180.
Parameter:	
number	- number in seconds.

system_sync=number

Meaning	- maximal difference between the **asix** system time and the DataPAF counter time, after which a system time synchronization with the counter time will occur. If the parameter value is 0, then the system time is not synchronized with the counter time.
Default value	- 0.
Parameter:	
number	- number in seconds.

period_Check=number

Meaning	- interval, with which the change of current calculation period of the counter is checked.
Default value	- 180.
Parameter:	
number	- number in seconds.

max_history=number

Meaning	- determines a time period from the current moment backwards, for which historical data, saved in station memory, will be read.
Default value	- 35.
Parameter:	
number	- number in days.

history_Buffer_Removal=number

Meaning	- the parameter determines a time period, after which buffers containing historical data, read for needs of an archiving module, are removed.
Default value	- 120.
Parameter:	
number	- the time is given in minutes.

CRC16=YES/NO

Meaning	- determines if the CRC16 validity check has to be used in the communication with the counter. If NO is given, then the sum of sequent bytes will be calculated.
Default value	- YES.

log=file_name

Meaning	- the parameter determines the name of file, to which additional diagnostic information will be written.
Default value	- lack.

alarm_Code =number

Meaning	- the parameter determines the number of an alarm, generated by the driver in case of loss and re-establishing of connection with the station. The value of -1 (default) causes that alarms are not generated. In a situation of a connection loss, a number specifying the cause of connection loss is transferred together with the alarm code: 0 – complete lack of any answer from the station;

	1 – timeout;
	2 – line errors (frame, parity, overrun errors);
	3 – checksum errors;
	4,... - other errors.
	This number determines the end status of the last attempt of connection establishing.
Default value	- lack.

☑ ***energy_error =number***

Meaning	- determines situations, when the status of the XEN variable value assumes an error value and it is the sum of the following values: 1 error when a short decay of voltage; 2 error when a long decay of voltage; 4 error when a pause in input line of the impulse counter.
Default value	- 0.

☑ ***mult =number***

Meaning	- defines a value of all input multipliers. Input multipliers are used for calculating the energy on the basis of the impulse number. If this parameter is given, then the driver will not read values of multipliers from the energy counter. If beside the parameter *mult*, parameters determining values of multipliers for individual channels (*multn*) are also used, then the parameter *mult* must be placed before all parameters *multn*.
Default value	- lack.
Parameter:	
number	- the multiplier is a floating-point number.

☑ ***multn =number***

Meaning	- defines an input multiplier value for a channel with the number *n* (1-31). Input multipliers are used for calculating the energy on the basis of the impulse number. If this parameter is given, then the driver will not read values of multipliers from the energy counter. If the parameter *mult* is also given, then it must be placed before all parameters *multn*.
Default value	- lack.
Parameter:	
number	- the multiplier is a floating-point number.

EXAMPLE

```
[datapaf]
....
;definition of multiplier for all channels:
mult = 2.7
;definition of multiplier for channel 6:
mult6 = 3.3
```

Driver Parameterization Examples

EXAMPLE

```
[ASMEN]
.....
KOMIN 2= DataPAF ,COM2
....

[DataPAF:2]
baud=19200
Numer=4800
```

In the above example the station named KOMIN 2, connected to the port COM2, is defined. The transmission speed of 19200 bps will be used.

In case of time synchronization of the **asix** station with the time of a chosen counter – the definition of this time should be placed in the ini section for the given counter. For this purpose the parameter *system_sync*, to which the value of allowed difference of times in seconds, is assigned.

EXAMPLE

```
[ASMEN]
...
CHAN1=DataPAF,COM2
KANAL2=DataPAF,COM4
...
[CHAN1]
system_sync=5
log=DATAPAF.log
diagnostics=14
```

In the example above the system time will be synchronized with the time of an energy counter connected to the port COM2. By placing the records *diagnostics=14* and *log=DATAPAF.log* in the counter section, it is possible to receive the record of time synchronization diagnostics in the file *DATAPAF.log*. The time is synchronized with an accuracy of 1 second.

Definition of Variables

EXAMPLE

An example of the ASMEN variable definition:

3010E15m1, DATAPAF Średnia 15min energii kan. 1, XEN1, CHAN1, 1, 60, NOTHING_FP
3010EBDSH, DATAPAF suma energii biernej kanałów w pop.okres, PEN11.8, CHAN1, 1, 3600, NOTHING_FP

The List of All the Variable Types Supported by the DataPAF Driver

c – signifies the number of channel 1..30
s – time zone 0..8

Table 7. The List of All the Variable Types Supported by the DataPAF Driver.

Name	Type
ENc.s	FLOAT
PENc.s	FLOAT
MMc.s	FLOAT
PMMc.s	FLOAT
MMIc	FLOAT
IMIc	WORD
IMPc	WORD
BRKc	BYTE
POFc	BYTE
XENc	FLOAT
IMTc	WORD
Lc	FLOAT
PLc	FLOAT
TIM	TEXT
DAT	TEXT
DTIM	TEXT
ALRc	BYTE
GALR	DWORD
INMULc	FLOAT
INMULCNTc	WORD
INMULKUKIc	FLOAT
INMULCONSTc	DWORD
INMULDIGc	WORD
MCHAN	WORD
PNT	WORD
REG	WORD
COL	WORD
CHAN	WORD
SUM	WORD
INTEG	WORD
PPS	DWORD
PPE	DWORD
CPS	DWORD
CPE	DWORD

Historical Data

Historical data are available for the following variable types: POF, IMP, BRK and XEN.

1.15. DDE Driver

Driver Use

The driver DDE is used to define the channel of the ASMEN module referring the variables shared by the driver of industrial controller implemented as a DDE server (called further shortly a DDE server).

In the new version of the driver while defining the ASMEN module channel, besides a DDE server name, also a name of the service registered by the DDE server and a DDE connection name have to be declared. Thanks to association of a DDE connection with a channel instead of with each variables separately, the way of variable definition is significantly simplified and the definitions of variables are more legible.

Supporting the variable groups allows the DDE driver to receive data from a DDE server in form of groups of variables and not only in form of single variables. Thanks to such solution the performance of data transmission may increase enormously. The maximal transmission performance by using the DDE protocol amounts up to 150 single-element groups of variables per second (at a computer with PCU Pentium 166 MHz) and descends slowly when the group size increases. If a group contains only one variable, then it is possible to transfer 150 variables per second whereas the group size amounts e.g. 25 variables, then it is possible to transfer 3000 variables per second.

A DDE server within each connection may make available a special variable, the value of which defines whether the variables retrieved from this connection have correct values, whether the connection with an industrial controller works correctly. The DDE driver assumes that such special variable has the *Status* name, but it is possible to define such variable separately for each DDE connection.

External Specification

The DDE driver is a dynamic link library (DLL) with an interface meeting requirements of the ASMEN module. ASMEN starts the driver after having found in the [ASMEN] section of application configuration file a definition of the channel referring to the DDE driver in the following form:

channel_name = DDE, *service*, *topic*

where:

channel_name	- channel name of the ASMEN module;
service	- service name registered by the DDE server;

topic	- topic name of a connection supported by the DDE server.

The names of connection service and topic are specific for each driver of the industrial controller implemented in form of a DDE server and their description may be found in the documentation of a given driver.

EXAMPLE

Examples of channel definitions.

KanAdam = DDE, Adam, E2018

The ASMEN channel named KanAdam is associated with a DDE driver and the topic of connection named E2018, made available by the DDE server named Adam.

KanGE = DDE, GESNP, GE

The ASMEN channel named KanGE is associated with the DDE driver and the topic of conenction named GE, shared by the DDE server named GENSP.

Variable Definitions

After the DDE driver start-up the ASMEN module transfers to the DDE driver an information about process variables taken from the definition file of variables. The definition of a variable used by the DDE driver is as follows:

asmen_variable, description, „item | *number_in_gorup* | *type", channel_name, quantity, refreshing_frequency, conversion_function*

where:

asmen_name	- variable name of the ASMEN module;
item	- name of a variable or a group of variables accessed by the DDE server, maximal length of this name amounts 255 characters (255 characters, it is the maximal length of text served by the DDE protocol);
number_in_group	- number of a variable in the group of variables to which the ASMEN variable refers; the enumeration begin from 1; if the DDE server as the *item* sends variables individually, then this parameter should have a value of 1;
type	- type of the ASMEN variable.

As the *type* parameter it may be given the letter:
S – short,
W – word,
L – long,
D – double word,
F – float.

The variable type definition is necessary to make possible the variable conversion to the binary form, used by the ASMEN module. The data from the DDE server are sent in text form and it is not possible to define their type unambiguously, therefore its type in variable definition is defined as public.

If a variable is to be not only read but also written, then the source of such variable in a DDE server must be a single variable. If to the ASMEN variable, which is not a part of variables group, you try to give a new value from the application level of **asix**, then an error occurs. If it were possible to give a new variable to only one variable in the group, the DDE driver while sending this group to the server should send also new values of the other variables in this group. The use of recently sent values of the other variables in the group as their new values might cause hard perturbations.

EXAMPLE

Examples of Variable Definitions

Z1, "O1, 1, W", KanAdam, 1, 1, NOTHING

The ASMEN variable Z1 is associated with the first variable in the variable group named O1 taken from the DDE server specified in the definition of the KanAdam channel.

Z2, "%I1, 5, s", KanalGE, 1, 1, NOTHING

The ASMEN variable Z2 is associated with the fifth variable in the variable group named %I1 taken from the DDE server specified in the definition of the KanGE channel.

DDE Section

The DDE section in the INI file of an **asix** application may contain the following entry.

☑ ***StatusVariable = service, topic, item***

Meaning	- the item of an initialization file, it defines the name of a variable handled by the DDE server. It allows to monitor the connection status of the DDE server with an industrial controller. If the value of this variable equals 1, the connection with the controller works correctly; if the variable value equals 0, the connection with the controller is broken. In order to define a status variable it is necessary in the initialization application file to define the item *StatusVariable* and to give to it a value in the format *service, topic, item*.
Default value	- by default, it is assumed that there is no status variable within the connection.

Parameter:

service and *topic*	- define the connection;
item	- is the name of status variable within this connection.

A DDE server may support many DDE connections simultaneously and for each of them a status variable may be defined separately.

EXAMPLE

Examples of definition of status variable:

```
[DDE]
StatusVariable = TPERM, S1, Connected
```

Within the connection TPERM, S1 the name of the status variable is Connected.

Thread Priority of the Driver

PriData – value of thread priority of the driver sending data to the ASMEN module, the default value is set to 1.

Internal Specification

The driver, at the first reference to the DDE server described by a pair of *service, topic*, establishes a connection with this server and keeps it on continuously up to ending the activity of the driver or of the DDE server. At the first reading of the first variable belonging to the variable group a refreshing of this variables group begins. The new values are written to the internal buffer of the driver and transferred to the ASMEN module in case of reading of refresh type. In case of ordinary reading the data are read directly from the DDE server. If it is not possible to begin refreshing, the variable value is read at each reference of the ASMEN module to it. Writing operations are carried out always synchronously and they update the internal buffer of the variable.

In case of breaking the connection with a DDE server, the DDE driver tries to rebuild this connection.

1.16. DP - Driver of PROFIBUS DP Network Protocol for PROFOboard Board

Driver Use

The DP driver is used for data exchange in the PROFIBUS network with devices operating according to the PROFIBUS DP standard. The **asix** system computer must be provided with the PROFIboard NT communication processor board and with the PROFIboard NT v 5.20 software package of Softing GmbH.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel working according to the DP protocol is given below:

logical_name=DP,*card_no, address,section_name*

where:

board_no	- number of the PROFIboard NT board which is used for communication with a given device (DP slave). In the present version the DP driver can control only one PROFIboard NT board;
address	- address assigned to the DP device;
section_name	- name of section in the application INI file, where the configuration parameters of the given DP device are contained.

The DP driver is loaded as a DLL automatically.

Configuration of DP Devices

Configuration parameters of each DP device (DP slave) should be declared in a separate section of the application INI file. These parameters should be accessible in the device specification or in the GSD file supplied by the manufacturer with the device. The section name is assigned by the user and must be unique in the application INI file.

The individual configuration parameters are transferred in separate items of the section. Each item is as follows:

parameter=value

where:

parameter	- type of configuration parameter;
value	- parameter value (in decimal format !)

Types of configuration parameters, which **must be transferred** by the application designer are given below:

IDENT_NUMBER	device identifier	(PNO ident. number)
GROUP_IDENT	group identifier	(group member bits)
USER_PRM_DATA		(prm_user_data)

.. variable number of USER_RPM_DATA items

.

USER_PRM_DATA	(prm_user_data)
MODULE_ID	(cfg_data)

.. variable number of MODULE_ID items

.

MODULE_ID	(cfg_data)

aat_data parameters are optional and if not declared they are set to 0.

NUMBER_OF_INPUTS	(aat_data)
NUMBER_OF_OUTPUTS	(aat_data)
OFFSET_OF_INPUTS	(aat_data)
OFFSET_OF_OUTPUTS	(aat_data)

slave_data parameters are optional and if not declared they are omitted.

SLAVE_DATA	(slave_user_data)

.. variable number of SLAVE_DATA items

.

SLAVE_DATA	(slave_user_data)

Addressing the Process Variables

The values transferred from modules connected to the DP device are written to an I/O buffer of the DP driver, in order resulting from arrangement of Input/Output modules in the cassette of the DP device. The addressing process variables requires an indication of:

- buffer type (input buffer or output buffer);
- byte no. (in the buffer), where the value of a given input/output is stored; depending on the type of process variable, the variable value occupies one byte (one-byte type variable) or 2 successive bytes (2-byte type variable);
- type of the variable (one-byte or 2-byte).

The syntax of symbolic address which is used for variables belonging to the DP driver channel is as follows:

<type><index>

where:

type	- type of process variables:
IB	- byte from the input buffer,
IW	- 2 successive bytes from the input buffer treated as a fixed-point unsigned number in INTEL format,
IDW	- 4 successive bytes from the input buffer treated as a double word in INTEL format,
IFP	- 4 successive bytes from the input buffer treated as a floating-point number in INTEL format,

IWM	- 2 successive bytes from the input buffer treated as a fixed-point unsigned number in MOTOROLA format,
IDWM	- 4 successive bytes from the input buffer treated as a double word in MOTOROLA format,
IFPM	- 4 successive bytes from the input buffer treated as a floating-point number in MOTOROLA format,
OB	- byte from the output buffer,
OW	- 2 successive bytes from the output buffer treated as a fixed-point unsigned number in INTEL format,
ODW	- 4 successive bytes from the output buffer treated as a double word in INTEL format,
OFP	- 4 successive bytes from the output buffer treated as a floating-point number in INTEL format,
OWM	- 2 successive bytes from the output buffer treated as a fixed-point unsigned number in MOTOROLA format,
ODWM	- 4 successive bytes from the output buffer treated as a double word in MOTOROLA format,
OFPM	- 4 successive bytes from the output buffer treated as a floating-point number in MOTOROLA format;
Index	- number of the byte in the input/output buffer.

EXAMPLE

IB9	- 9-th byte from the area of inputs
IW2	- word created from the 2-nd and 3-rd byte of the input area (INTEL format)
IWM2	- word created from the 3-rd and 2-nd byte of the input area (MOTOROLA format)
IDW5	- double word created from the 5-th, 6-th, 7-th and 8-th byte of the input area (INTEL format)
IDWM5	- double word created from the 8-th, 7-th, 6-th and 5-th byte of the input area (MOTOROLA format)

Driver Configuration

The DP protocol driver may be configured with use of information contained in **[DP]** section placed in the application INI file. Items in the DP section have the following syntax:

item_name = [number [,number]] [text][YES|NO]

☑ ***TRANSMISSION_SPEED=network_no,baud_id***

Meaning	- the item is used to declare the transmission speed in the PROFIBUS DP network.
Default value	- by default, the transmission speed is set to 1,5 MB.
Parameter:	
network_no	- number of the PROFIBUS DP network (in the present version always set to 1),
baud_id	- identifier of transmission speed in a PROFIBUS DP network:

0 -	9,6 kB
1 -	19,2 kB
2 -	93,75 kB
3 -	187,5 kB
4 -	500 kB

5 -	750 kB
6 -	1,5 MB
7 -	3 MB
8 -	6 MB
9 -	12 MB
11 -	45,45 kB

EXAMPLE

A declaration of transmission speed of 750 kB:

```
TRANSMISSION_SPEED = 1, 5
```

☑ ***REFRESH_CYCLE=number***

Meaning	- the item is designed to declare an interval between successive data readings from buffers of the PROFIboard NT board to the structures of the DP driver.
Default value	- by default, the DP driver reads data from buffers of the PROFIboard NT board every 0.5 second.
Parameter:	
number	- number of 0.5-second intervals, which must pass between successive data readings from buffers of the PROFIboard NT board.

EXAMPLE

A declaration of data reading every 1 second:

```
REFRESH_CYCLE=2
```

☑ ***CONSOLE=YES|NO***

Meaning	- the item allows creating a console window where the DP driver messages, concerning the status of communication between an **asix** system computer and DP devices, are displayed.
Default value	- by default, the console window is not created.

☑ ***LOG_FILE=file_name***

Meaning	- the item allows to define a file where all messages of the DP driver, concerning to the status of communication between an **asix** system computer and DP devices, will be written. If the item does not define the full path, then the log file is created in the current directory.
Default value	- by default, the log file is not created.

EXAMPLE

An example item declaring a transmission channel using the DP protocol for the communication with ET200U no. 7 unit is given below. The following input/output boards are connected to the ET200U (in order of their arrangement in the list):

- Digital Output module (8 outputs) 6ES5 461-8MA11 (identifier 32),
- Analog Input module (4 inputs) 6ES5 464-8ME11 (identifier 83),
- Digital Input module (8 inputs) 6ES5 431-8MA11(identifier 16).
- Digital Output module (8 outputs) 6ES5 461-8MA11 (identifier 32).
- *CHAN1=DP,1,7,SIEM8008*

The transmission channel named CHAN1 has the following parameters defined:

- DP protocol,
- communication by means of the PROFIboard NT board with the number of 1,
- DP device has number 7 in the PROFIBUS DP network,
- configuration parameters of DP device are contained in the section SIEM8008.

The content of the section [SIEM8008] defining the sample configuration of the ET200U is as follows (all the values are decimal):

```
[SIEM8008]
IDENT_NR=32776
GROUP_IDENT=0
USER_PRM_DATA=0
MODULE_ID=32
MODULE_ID=83
MODULE_ID=32
MODULE_ID=16
```

In the configuration under consideration the area of inputs has 9 bytes. The meaning of the bytes is as follows:

bytes 1,2	- analog input 1
bytes 3,4	- analog input 2
bytes 5,6	- analog input 3
bytes 7,8	- analog input 4
byte 9	- digital input byte

In the configuration under consideration the area of outputs has 2 bytes. The meaning of the bytes is as follows:

byte 1	- digital output byte (the first module 6ES5 451-8MA11)
byte 2	- digital output byte (the second module 6ES5 451-8MA11)

EXAMPLE

Example declarations of process variables are given below:

```
# X1 – digital output – 1-st byte of output buffer
X1, OB1,      CHAN1,   1,  1, NOTHING_BYTE
# X2 – digital output  - 2-nd byte of output buffer
X2, OB2,      CHAN1,   1,  1, NOTHING_BYTE
# X3 – digital input  - 9-th byte of input buffer
X3, IB9,      CHAN1,   1,  1, NOTHING_BYTE
# X4 – analog input 1 - 1-st and 2-nd bytes of input buffer
X4, IW1,      CHAN1,   1,  1, NOTHING
# X5 – analog input 2 - 3-rd and  4-th byte of input buffer
X5, IW3,      CHAN1,   1,  1, NOTHING
# X 6 – analog input 3 - 5-th and 6-th byte of input buffer
X6, IW5,      CHAN1,   1,  1, NOTHING
```

```
# X7 – analog input 4 - 7-th and 8-th byte of input buffer
X7, IW7,       CHAN1,   1,  1, NOTHING
```

1.17. DP5412 - Driver of PROFIBUS DP Protocol for CP5412 Card

Driver Use

The DP5412 driver is used for data exchange in the PROFIBUS network with devices operating according to the PROFIBUS DP standard. The CP5412(A2) communication processor card and the DP-5412 software package (version 4.1 or higher) or the CP5613 card with the SIEMENS DP-5613 package must be installed on the **asix** system computer.

Declaration of Transmission Channel

The full syntax of item declaring the transmission channel working according to the DP5412 protocol is given below:

logical_name=DP5412, *card_no, address*

where:

board_no	- number of the CP5412 (A2) or CP5613 card used for communication with the given DP device (DP slave). In the present version the DP5412 driver may handle only one CP5412(A2) or CP5613 card,
address	- address assigned to the DP device.

The DP5412 driver is loaded as a DLL.

Configuration of DP Devices

The DP devices (DP slave) configuration is performed by the COM PROFIBUS program included in the DP-5412 package.

The application designer must ensure the compatibility of numbers assigned to the DP devices during the DP network configuration with the program COM PROFIBUS and of DP device numbers declared in the **asix** application INI file.

Addressing the Process Variables

The values transferred from modules connected to the DP device are written to an input buffer and to an output buffer of the DP5412 driver, in order according to the arrangement of Input/Output modules in the DP device cassette. The addressing the process variables consists in indication of:

- buffer type (input buffer or output buffer);
- byte no. (in the buffer), where the value of a given input/output is stored; depending on the type of process variable, the variable value occupies one byte (one-byte type variable) or 2 successive bytes (2-byte type variable);
- type of the variable (one-byte or 2-byte).

The syntax of symbolic address which is used for variables belonging to the DP5412 driver channel is as follows:

<type><index>

where:

type		- type of process variables:
	IB	- byte from the input buffer,
	IW	- 2 successive bytes from the input buffer treated as a fixed-point unsigned number in INTEL format,
	IDW	- 4 successive bytes from the input buffer treated as a double word in INTEL format,
	IFP	- 4 successive bytes from the input buffer treated as a floating-point number in INTEL format,
	IWM	- 2 successive bytes from the input buffer treated as a fixed-point unsigned number in MOTOROLA format,
	IDWM	- 4 successive bytes from the input buffer treated as a double word in MOTOROLA format,
	IFPM	- 4 successive bytes from the input buffer treated as a floating-point number in MOTOROLA format,
	OB	- byte from the output buffer,
	OW	- 2 successive bytes from the output buffer treated as a fixed-point unsigned number in INTEL format,
	ODW	- 4 successive bytes from the output buffer treated as a double word in INTEL format,
	OFP	- 4 successive bytes from the output buffer treated as a floating-point number in INTEL format,
	OWM	- 2 successive bytes from the output buffer treated as a fixed-point unsigned number in MOTOROLA format,
	ODWM	- 4 successive bytes from the output buffer treated as a double word in MOTOROLA format,
	OFPM	- 4 successive bytes from the output buffer treated as a floating-point number in MOTOROLA format;
Index		- number of the byte in the input/output buffer.

EXAMPLE

IB9	- 9-th byte from the area of inputs
IW2	- word created from the 2-nd and 3-rd byte of the area of inputs (INTEL format)
IWM2	- word created from the 3-rd and 2-nd byte of the area of inputs (MOTOROLA format)
IDW5	- double word created from the 5-th, 6-th, 7-th and 8-th byte of the area of inputs (INTEL format)
IDWM5	- double word created from the 8-th, 7-th, 6-th and 5-th byte of the area of inputs (MOTOROLA format)

Driver Configuration

The DP5412 driver may be configured using the **[DP5412]** section placed in the application INI file. Items in the DP5412 section have the following syntax:

item_name = [number [,number]] [text][YES|NO]

☑ ***REFRESH_CYCLE=number***

Meaning	- the item used to declare an interval between successive data readings from buffers of the CP5412(A2) or CP5613 CARD to the DP5412 driver structures.
Default value	- by default, the DP5412 driver reads data from buffers of the CP5412(A2) or CP5613 card every 0.5 second.
Parameter:	
number	- number of 0.5-second intervals, which must pass between successive data readings from buffers of the CP5412 (A2) or CP5613 card.

A declaration of data reading every 1 second:

```
REFRESH_CYCLE=2
```

☑ ***CONSOLE=YES|NO***

Meaning	- the item allows to create a console window, where the DP5412 driver messages, concerning to status of communication between an **asix** system computer and DP devices, are displayed.
Default value	- by default, the console window is not created.

☑ ***LOG_FILE=file_name***

Meaning	- the item allows to define a file where all the DP5412 driver messages, concerning to the status of communication between an **asix** system computer and DP devices, will be written. If the item does not define the full path, then the log file is created in the current directory.
Default value	- by default, the log file is not created.

EXAMPLE

An example item declaring a transmission channel using the DP5412 protocol for the communication with ET200U no. 7 is given below. The following input/output modules are connected to the ET200U (in order of their arrangement in the list):

- Digital Output module (8 outputs) 6ES5 461-8MA11,
- Analog Input module (4 inputs) 6ES5 464-8ME11,
- Digital Input module (8 inputs) 6ES5 431-8MA11.
- Digital Output module (8 outputs) 6ES5 461-8MA11.

```
CHAN1=DP5412,1,7
```

The transmission channel named CHAN1 has the following parameters defined:

- DP5412 protocol,
- communication via the CP5412 (A2) card no. 1,
- DP device is assigned no. 7 in the PROFIBUS DP network.

In the considered configuration the area of inputs has 9 bytes. The meaning of the bytes is as follows:

bytes 1,2	- analog input 1 (module 6ES5 431-8ME11),
bytes 3,4	- analog input 2 (module 6ES5 431-8ME11),
bytes 5,6	- analog input 3 (module 6ES5 431-8ME11),
bytes 7,8	- analog input 4 (module 6ES5 431-8ME11),
byte 9	- digital input byte (module 6ES5 431-8MA11).

In the considered configuration the area of outputs has 2 bytes. The meaning of the bytes is as follows:

byte 1	- digital output byte	(the first module 6ES5 451-8MA11)
byte 2	- digital output byte	(the second module 6ES5 451-8MA11)

Exemplary declarations of process variables are given below:

```
# X1 – digital output – 1-st byte of output buffer
X1,     OB1,  CHAN1,  1,  1, NOTHING_BYTE
# X2 – digital output  - 2-nd byte of output buffer
X2,     OB2,  CHAN1,  1,  1, NOTHING_BYTE
# X3 – digital input  - 9-th byte of input buffer
X3,     IB9,  CHAN1,  1,  1, NOTHING_BYTE
# X4 – analog input 1 - 1-st and 2-nd bytes of input buffer
X4,     IW1,  CHAN1,  1,  1, NOTHING
# X5 – analog input 2 - 3-rd and  4-th byte of input buffer
X5,     IW3,  CHAN1,  1,  1, NOTHING
# X 6– analog input 3 - 5-th and 6-th byte of input buffer
X6,     IW5,  CHAN1,  1,  1, NOTHING
# X7 – analog input 4 - 7-th and 8-th byte of input buffer
X7,     IW7,  CHAN1,  1,  1, NOTHING
```

1.18. DMS285 - Driver of Protocol for DURAG DMS 285 Analyzers

Driver Use

The DMS285 protocol driver is designed to establish the communication between an **asix** system computer and a DURAG D-MS285 computer for emission monitoring. The driver operates with devices supporting the protocol of version 1.22.

Declaration of Transmission Channel

A logical channel is a logical connection of a computer and the DMS285 station. The logical channel is defined with use of an appropriate record in the [ASMEN] section.

The full syntax of the item declaring the transmission channel working according to the DMS285 protocol is given below:

logical_name=DMS285,*COMn*

where:

COMn	- number of the serial port to which the network of DMS285 controllers is connected.

Driver Configuration

Each defined channel may have its own section, the name of which is the logical name of the channel. The given COMn port may also have its own section named **[DMS285:n]**. The values defined in such section become default ones for particular stations. The default values for particular serial ports are retrieved from the section named **[DMS285]**. Parameters of transmission via a serial interface can't be placed in the sections concerning particular stations, i.e. they may be placed only in the [DMS285] and [DMS285:n] sections.

☑ ***name=station_name***

Meaning	- name of the DMS285 station.
Default value	- by default, a logical channel name is assumed as a station name.

Parameter:	
Station_name	- name is completed with spaces to the length of 8 characters.

☑ ***baud =number***

☑ ***bps=number***

Meaning	- determines transmission speed.
Default value	- 9600.
Parameter:	
number	- number in Bd.

☑ ***parity=parity_parameter***

Meaning	- determines parity.
Default value	- e.
Parameter:	
parity_parameter	- default value:

n	- no parity bit,
o	- odd parity check,
e	- even parity check,
m	- mark
s	- space.

☑ ***stop_bits =number***

Meaning	- determines number of stop bits.
Default value	- 1.
Parameter:	
number	- admissable values are 1 and 2.

☑ ***word =number***

word_length=number

Meaning	- word length.
Default value	-8.
Parameter:	
number	- allowed values are from 5 to 8.

☑ ***time_out =number***

☑ ***timeout=number***

Meaning	- waiting time for answer from DMS285A.
Default value	- 10.
Parameter:	
number	- number in seconds.

Bad_Data_Classes = class1, class2, ..., classN

Meaning	- parameter determines numbers of classes, which cause invalidity of data 44.5K.n and 44.5M.n. The class number is taken up from the variable 44.1.n (Klassenangabe der Konz.).
Default value	- lack (-) (class value has no influence on the validity of read data).
Parameter:	
class1, class2, ..., classN	- numbers of classes.

Bad_Data_Status = status1, status2, ..., statusN

Meaning	- parameter determines values of the variable 43.3.n (Zustand des Kanals – Channel state), for which values of variables 43.7K.n, 43.7M.n, 43.9K.n, 43.9M.n, 43.10K.n and 43.10M.n are assumed as invalid.
Default value	- lack (-) (value of the variable 43.3.n has no influence on validity of read data).
Parameter:	
status1, status2, ...,statusN	- values of the variable 43.3.n (Zustand des Kanals).

Bad_Data_Classes2 = class1, class2, ..., classN

Meaning	- parameter determines numbers of classes, invalidity of data 44.5K.n and 44.5M.n while calculating 48-hour averages. The class number is taken up from the variable 44.1.n (Klassenangabe der Konz.).
Default value	- lack (-) (class value has no influence on the validity of read data).
Parameter:	
class1, class2, ..., classN	- numbers of classes.

Minimal_Measurements =number

Meaning	- minimal number of measurements required for calculation of a 48-hour average.
Default value	- lack.
Parameter:	
number	- number of measurements.

Archive_type =number

Meaning	- parameter was deleted in version 1.12.
Default value	- B.
Parameter:	
number	- symbol of archive type.

☑ ***Max_Time_Difference =number***

Meaning	- determines the maximal difference in seconds between the **asix** system time and DMS295 station time, after exceeding of which warnings will be output to *'Control Panel'*. Station time is read only during reading of 43 and 50 type variables.
Default value	- 60.
Parameter:	
number	- number in seconds.

☑ ***Var_44_1_n =variable_name,channel_number***

Meaning	- name of the variable 44.1.n and archive type, which is saved in the archive and used simultaneously for calculation of 48-hour average 44.105K.n. After the variable name the archive type should be given after a comma. *n* signifies the channel number.
Default value	- lack.
Parameter:	
variable_name	- determines the name of 44.1.n variable; n signifies the channel number.
channel_number	- number of the channel.

EXAMPLE

Var_44_1_3 = K3_H_KLAS_Pyl, M.

NOTE *While configuring the variable for ASPAD the option DO_NOT_PACK should be* ***absolutely*** *given!!!*

☑ ***Var_44_5K_n =variable_name,channel_number***

Meaning	- name of the variable 44.5K.n and archive type, which is saved in the archive and used simultaneously for calculation of 48-hours average 44.105K.n. After the variable name the archive type should be given after a comma. *n* signifies the channel number.
Default value	- lack.
Parameter:	
variable_name	- determines the name of 44.5K.n variable; *n* signifies the channel number.
channel_number	- number of the channel.

EXAMPLE

Var_44_5K_3 = K3_H_KONC_Pyl, M.

NOTE *While configuring the variable for ASPAD the option DO_NOT_PACK should be* ***absolutely*** *given!!!*

☑ *Auto_sync = number*

☑ *Autosync=number*

Meaning	- determines the maximal difference between the **asix** system time and the DMS285 station time, after exceeding of which the driver will set the station time on the **asix** system time. Time synchronization occurs only during data reading from the station. The minimal value of this parameter equals 10 seconds. If a smaller value is given, then 10 seconds will be assumed.
Default value	- no time synchronization.
Parameter:	
number	- number in seconds.

☑ *Max_auto_sync = number*

☑ *MaxAutoSync= number*

Meaning	- determines the maximal value between the **asix** system time and the DMS285 station time, after exceeding of which the driver will **not** synchronise the station time even if this difference exceeds the value determined by the *AutoSync* parameter.
Default value	- 6800.
Parameter:	
number	- number in seconds.

☑ *Log = file_name*

Meaning	- parameter determines a file name, to which additional diagnostic information will be written. Parameter may be placed only in [DMS285:n] and [DMS285] sections.
Default value	- lack.

☑ *Time_limiter = number*

Meaning	- determines the maximal deviation of data time in hours in relation to the current system time, whose exceeding causes a data is found invalid. The time of a data read from DURAG computer must be contained in the time interval current time +- deviation, so that it might be found valid. If the parameter has a value of 0 then data time is not checked.
Default value	- 24.
Parameter:	
number	- number in hours.

☑ ***Alarm_Code = alarm_number***

Meaning	- parameter determines the number of alarm generated by the driver in case of loss and re-establishing the connection with the station. The value of -1 (default) causes that the alarms are not generated. In case of connection loss, one of the following number specifying the reason for the dropped connection is relayed with the alarm code: 0 - complete lack of any answer from the station, 1 - timeout, 2 - errors of lines (errors of border, parity, overrun), 3 - errors of checksum, 4 - other errors. This number determines the status of the end of last attempt made to establish a connection.
Default value	- lack.
Parameter:	
Alarm_number	- alarm number.

☑ ***Max_history =number***

Meaning	- determines a time period, counted from the current moment backwards, for which historical data, stored in the station memory, will be read.
Default value	- 20.
Parameter:	
number	- number in days.

Examples of Driver Configuration

EXAMPLE 1

```
[ASMEN]
.....
KOMIN 2=DMS285,COM2
....


[DMS285:2]
baud=19200
```

In the above example a station named KOMIN 2 is connected to the COM2 port. The transmission speed of 19200 bps will be used.

EXAMPLE 2

```
[ASMEN]
.....
KOMIN 1=DMS285,COM2
KOMIN 2=DMS285,COM2
KOMIN 3=DMS285,COM2
KOMIN 4=DMS285,COM3
KOMIN 5=DMS285,COM3
KOMIN 6=DMS285,COM4
....


[DMS285]
;Default values for all stations
speed=19200
Invalidity_Status= 1, 6, 14

[DMS285:3]
;Default values for stations connected to the COM3 port
speed=9600

[KOMIN 2]
Invalidity_Status= 5

[KOMIN 4]
Invalidity_Status= -
```

In the above example the stations of names from KOMIN 1 up to KOMIN 6 are defined. The stations KOMIN 1, KOMIN 2 and KOMIN 3 are connected to the COM2 port. The stations KOMIN 4 and KOMIN 5 are connected to the COM3 port. The station KOMIN 6 is connected to the COM4 port. All the serial ports except COM3 will work with a speed of 19200 baud. The COM3 port will work with a speed of 9600 baud. All the stations except the stations KOMIN 2 and KOMIN 4 will use invalidity statuses 1, 6 and 14. The station KOMIN 2 uses a value of 5 as an invalidity status. The station KOMIN 4 does not use any invalidity status – setting parameter „-„ was necessary to change the default values set in the [DMS285] section.

Defining the Process Variables

The variable definition is based on the DMS285 protocol description. The list of all the types of variables is given at the end of this chapter.

$$\text{type.subtype}\begin{bmatrix}\text{K}\\\text{M}\end{bmatrix}[\text{idx}]\begin{bmatrix}\text{.Mxx}\\\text{.subfield}\end{bmatrix}[\text{.channel}]$$

where:

type	- defines information type e.g.: 43 - instantaneous values, 44 - integrals, 17 - parameters;
subtype	- number of given information, e.g. 43 for instantaneous values defines actual current intensity in a given channel;
K,P.	- concentation/flow (only if the subtype contains data for both these categories);
idx	- index - only for indexed variables, e.g. classification; index is a number >= 1;
channel	- channel number; a channel number may be given only for the values related to the channel; in case of general information it should be omitted.
Mxx	- bit mask; *xx* is a number in hexadecimal code; on the datum received from the station, the AND operation with the number *xx* is executed;

subfield	- subfield name; for data representing time the following subfields are defined: SEC, MIN, HOUR, DAY, MONTH, YEAR - DWORD type, TIME, DATE, DATETIME - TEXT type.

In order to display values TIME, DATE, DATETIME the object STRING may be applied. The function NOTHING_TEXT must be use as the conversion function. The length of displayed string is given as the counter of elements.

EXAMPLE

An example of variable declaration:

ACT_TIME, actual date and time of the station, 43.1.DATETIME.1, CHANNEL-DMS285, 20,30, NOTHING_TEXT

Format of date and time:

dd-mm-rrrr gg:mm:ss.

EXAMPLE

43.11.1	- actual instantaneous value for the analog channel 1
43.1.DAY	- actual time of the station - month day number
43.7K.1	- actual concentration for the channel 1
43.2[2]	- digital inputs 16-23

Time of Data

The data are transferred by the driver to the **asix** system together with the time of their reception (time of the DMS285 station). In case of other types than the type 43, which does NOT contain actual time, the time is established on the ground of previously received packet of the type 43 (variable 43.1) and time of its reading. The time defined by the variable 44.20 is assigned to the variables 44,1, 44.5K.n and 44.5M.n. The time of the variables of the packet 48 is rounded down to the hour 00:00. The variables of the packet 48 arriving at 00:00 +- 2 min are treated as invalid. The variables 44.105M.n and 44.105K.n have always the time 00:00.

Historical Data

For the variables 44.5K and 44.5M it is possible to read historical data.

List of All the Variable Types Supported by DMS285 Driver

See the following tables.

Table 8. List of All the Variable Types Supported by DMS285 Driver.

Name	Type
0.1.n	FLOAT
1.1.n	FLOAT
2.1.n	FLOAT
3.1.n	FLOAT
4.1.n	FLOAT
5.1.n	FLOAT
6.1.n	WORD
6.2.n	WORD
6.3.n	WORD
6.4.n	WORD
6.5K.n	FLOAT
6.5M.n	FLOAT
7.1.n	FLOAT
8.1.n	FLOAT
9.1.n	FLOAT
10.1.n	FLOAT
11.1.n	FLOAT
12.1.n	FLOAT
13.1.n	FLOAT
14.1.n	FLOAT
15.1.n	WORD
16.1.n	DWORD
c – signifies the channel number 1..32 n – signifies the size number (schadstoff) 1..15	

Table 9. List of All the Variable Types Supported by DMS285 Driver (continuation)

Station Parametrieren	
17.1.n	FLOAT
: **Fehlerzustand** (x - fehlt)	
18.1.x	WORD
18.2.x	WORD
18.3.x	WORD
18.4.x	WORD
18.5.x	WORD
19.1[x].n	WORD
20.1[x].n	WORD
21.1[1].n - 21.1[3].n	FLOAT
22.1[1].n - 22.1[3].n	FLOAT
23.1[1] - 23.1[16]	BYTE
38.1.n	FLOAT
38.2.n	FLOAT
39.1.n	FLOAT
39.2.n	FLOAT
43.1	WORD
43.2[1] - 43.2.[8]	BYTE
43.3.n	BYTE
43.4.n	FLOAT
43.5.n	FLOAT
43.6.n	FLOAT
43.7K.n	FLOAT
43.7M.n	FLOAT
43.8K.n	FLOAT
43.8M.n	FLOAT
43.9K.n	FLOAT
43.9M.n	FLOAT
43.10.c	FLOAT
43.11.c	FLOAT
43.12.n	WORD
43.13.n	WORD
43.14.n	WORD
43.15.n	WORD
43.16.n	BYTE
43.17[1] - 43.17[18]	BYTE
43.18K.n	FLOAT
43.18M.n	FLOAT
43.19K.n,	FLOAT
43.19M.n	FLOAT

Table 10. List of All the Variable Types Supported by DMS285 Driver (continuation).

Aktueller Stand der Klasseninhalte in Binar	
46.4.n	WORD
46.5.n	DWORD
46.6.n	DWORD
46.8[1].n - 48.8[3].n	WORD
46.9.n	WORD
46.10[1].n - 48.10[21].n	WORD
46.11.n	WORD
46.12.n	WORD
46.13.n	WORD
46.14.n	WORD
46.15.n	WORD
46.16[1].n - 48.16[22].n	WORD
46.17.n	WORD
46.18.n	WORD
46.19.n	WORD
46.20.n	WORD
46.21.n	WORD
46.22.n	WORD
46.23.n	WORD
46.24.n	WORD
46.25.n	WORD
46.26.n	WORD
46.27[1].n - 48.27[21].n	WORD
46.28.n	WORD
46.29.n	WORD
46.30.n	WORD
46.31.n	WORD
46.32.n	WORD
46.33[1].n - 48.33[22].n	WORD
46.34.n	WORD
46.35.n	WORD
46.36.n	WORD
46.37.n	WORD
46.38.n	WORD
46.39.n	FLOAT
46.40.n	FLOAT
46.41.n	FLOAT
46.42.n	FLOAT
46.43.n	FLOAT

Table 11. List of All the Variable Types Supported by DMS285 Driver (continuation).

44.1.n,	BYTE
44.2.n,	FLOAT
44.3.n,	FLOAT
44.4.n,	FLOAT
44.5K.n,	FLOAT
44.5M.n,	FLOAT
44.6K.n,	FLOAT
44.6M.n,	FLOAT
44.7K.n,	FLOAT
44.7M.n,	FLOAT
44.8.n	FLOAT
44.9.n	DWORD
44.10.c	FLOAT
44.11[1] - 44.11[4]	BYTE
44.12K.n	FLOAT
44.12M.n	FLOAT
44.13K.n	FLOAT
44.13M.n	FLOAT
44.14K.n	FLOAT
44.14M.n	FLOAT
44.15[1] - 44.15[8]	BYTE
44.16K.n	FLOAT
44.16M.n	FLOAT
44.17.n	BYTE
44.18.n	BYTE
44.19K.n	BYTE
44.19M.n	BYTE
44.20	WORD
44.21[1] - 44.21[4]	BYTE
44.105K.n,	FLOAT
44.105M.n,	FLOAT
45.1.n	FLOAT
45.2.n	FLOAT
45.3.n	FLOAT
45.4.n	FLOAT
45.5.n	FLOAT
45.6.n	FLOAT
45.7.n	FLOAT
45.8.n	WORD

Table 12. List of All the Variable Types Supported by DMS285 Driver (continuation).

Klasseninhalte beim lt. Tageswechsel in Binar	
47.4.n	WORD
47.5.n	DWORD
47.6.n	DWORD
47.8[1].n - 48.8[3].n	WORD
47.9.n	WORD
47.10[1].n - 48.10[21].n	WORD
47.11.n	WORD
47.12.n	WORD
47.13.n	WORD
47.14.n	WORD
47.15.n	WORD
47.16[1].n - 48.16[22].n	WORD
47.17.n	WORD
47.18.n	WORD
47.19.n	WORD
47.20.n	WORD
47.21.n	WORD
47.22.n	WORD
47.23.n	WORD
47.24.n	WORD
47.25.n	WORD
47.26.n	WORD
47.27[1].n - 48.27[21].n	WORD
47.28.n	WORD
47.29.n	WORD
47.30.n	WORD
47.31.n	WORD
47.32.n	WORD
47.33[1].n - 48.33[22].n	WORD
47.34.n	WORD
47.35.n	WORD
47.36.n	WORD
47.37.n	WORD
47.38.n	WORD
47.39.n	FLOAT
47.40.n	FLOAT
47.41.n	FLOAT
47.42.n	FLOAT
47.43.n	FLOAT

Table 13. List of All the Variable Types Supported by DMS285 Driver (continuation).

Gesamt-Klassinh. b. lt. Tagesw. in Binar	
48.4.n	WORD
48.5.n	DWORD
48.6.n	DWORD
48.8[1].n - 48.8[3].n	WORD
48.9.n	WORD
48.10[1].n - 48.10[21].n	WORD
48.11.n	WORD
48.12.n	WORD
48.13.n	WORD
48.14.n	WORD
48.15.n	WORD
48.16[1].n - 48.16[22].n	WORD
48.17.n	WORD
48.18.n	WORD
48.19.n	WORD
48.20.n	WORD
48.21.n	WORD
48.22.n	WORD
48.23.n	WORD
48.24.n	WORD
48.25.n	WORD
48.26.n	WORD
48.27[1].n - 48.27[21].n	WORD
48.28.n	WORD
48.29.n	WORD
48.30.n	WORD
48.31.n	WORD
48.32.n	WORD
48.33[1].n - 48.33[22].n	WORD
48.34.n	WORD
48.35.n	WORD
48.36.n	WORD
48.37.n	WORD
48.38.n	WORD
48.39.n	FLOAT
48.40.n	FLOAT
48.41.n	FLOAT
48.42.n	FLOAT
48.43.n	FLOAT

1.19. DMS500 - Driver of Protocol for DURAG DMS 500 Analyzers

Driver Use

The DMS500 driver is designed for data exchange between the D-MS500 emission computer and the **asix** system by using serial interfaces. The driver supports devices with implemented company software in versions DMS500 v. 1.23, 1.55, 1.59.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel operating according to the DMS500 protocol is given below:

logical_name=DMS500,*COMn*

where:

COMn	- number of the serial port to which the DURAG emission computer is connected.

The DMS500 driver is loaded as a DLL automatically.

Addressing the Process Variables

The definition of process variables is based on the DMS protocol description titled *"Beschreibung der Kommunikation D-EVA mit DMS500"*.

The syntax of symbolic address which is used for variables belonging to the DMS500 driver channel is as follows:

type.subtype [K|M] [idx] [.Mxx|.subfield] [.channel]

where:

type	- define the type of information: 43 - instantaneous values,

	44 - integrals, 45 - parameters, 46 - classification of analog inputs – current results, 47 - classification of analog inputs – yearly results, 48 - classification of analog inputs – daily results, 56 - classification of digital inputs – current results, 57 - classification of digital inputs – yearly results, 58 - classification of digital inputs – daily results;
subtype	- number of required information; e.g. 13 for instantaneous values defines actual current intensity in a given channel;
K,M	- concentration/flow (only when a subtype contains data for both these categories);
idx	- index - only for indexed variables, e.g. classification; an index is a number bigger or equal to 1;
Mxx	- bit mask; *xx* is a number in hexadecimal code; on the data received from a DMS computer the AND operation with the number of *xx* is executed;
Subfield	- subfield name; for time values the following subfields are defined: SEC, MIN, HOUR, DAY, MONTH, YEAR;
Channel	- number of the channel; the channel number may be given only for the variables related to the channel. In case of general data it should be omitted.

EXAMPLES

43.14.1	actual instantaneous value of current for the analog channel 1
43.1.DAY	actual time of DMS computer - number of month day
43.17K.1	actual concentration for the channel 1
46.4K[5].1	actual value of the class 5 (concentration) for the channel 1

REPORT Special Variable

The special variable **REPORT** is a pseudo-variable of the 16-bit word type. Writing a given value to the REPORT variable causes reading an appropriate report (defined by this value) form the DMS computer. The report will be sent to a place defined in the **REPORT** item.

Access to Historical Data

The DMS500 driver enables the ASPAD module to access to the following historical data:

44.9K	- class number for concentration,
44.9M	- class number for flow,
44.16K	- concentration value,
44.16M	- flow value.

Time of Data

The data are transferred to the **asix** system together with time of their retrieving. In case of the type 45, for which the data packet does **not** contain current time (field no. 1 or 10 for types 56,57,58), the time is determined on the ground of previously received packet provided with time and on the ground of the time of its reading. Data packets do not contain the time for the type 45.

Incorrect Data

For the data 43.14 and 43.15, the DMS computer transfers status data (43.13). If any bit (specified in the protocol description) will be set, then such data is treated as incorrect.

Driver Configuration

Each defined logical channel has its own section, the name of which must be the same as the logical channel name. The ***COMn*** port, used by the given logical channel, may also have its own section named **[DMS500:n]**. Values defined in such section become the default values for particular DMS computers. The default values for particular serial interfaces are taken from the section named **[DMS500]**. Transmission parameters through a serial interface can’t be placed in sections concerning particular DMS computers.

Name=computer_name

Meaning	- the item is used to declare an 8-bit character name of the DMS computer. The name is completed with spaces up to 8-character length.
Default value	- by default, the name of the logical channel is assumed.

baud=number

Meaning	- the item is used to declare a transmission speed. The item may be used interchangeably with items: **bod**, **bps**.
Default value	- by default, the transmission speed is equal to 9600 Bd.
Parameter:	
number	- transmission speed in bauds.

parity=check_type

Meaning	- the item is used to declare the kind of parity check.
Default value	- by default, the parity check is even.
Parameter:	
check_type	- identifier of the kind of parity check:

n	- no parity bit,
o	- parity odd,
e	- parity even,
m	- mark,
s	- space.

stop=number

Meaning	- the item is used to declare a number of stop bits.
Default value	- by default, it is assumed to be 1 stop bit.
Parameter:	
number	- number of stop bits: 1 or 2.

word=number

Meaning	- the item is used to declare a number of bits in a transmitted character.
Default value	- by default, it is assumed that a character is of 8 bits in length.
Parameter:	
number	- number of bits in a character (5 to 8).

timeout=number

Meaning	- the item is used to declare waiting time for an answer from the DMS computer.
Default value	- by default, it is assumed 10 seconds.
Parameter:	
number	- waiting time for an answer in seconds.

Auto_sync=number

Meaning	- the item is used to declare an automatic synchronization of an **asix** system computer clock with a DMS computer clock. The parameter value determines maximal drift of time. The synchronization occurs when this drift limit is exceeded. The time from the DMS computer is received only during reading other data.
Default value	- by default, the time is not synchronized.
Parameter:	
number	- 0 (no synchronization) or maximal variance of times, (in seconds), after which the synchronization occurs.

History_Buffer_Removal=number

Meaning	- the item is used to declare the time, after which buffers containing historical data that were read for needs of ASPAD are removed.
Default value	- by default, history buffers are removed after 30 minutes.
Parameter:	
number	- time (in minutes), after which history buffers are removed.

Max_History_Buffers=number

Meaning	- the item is used to declare the maximal number of history buffers containing historical data that were read for needs of ASPAD. One buffer contains historical data for one channel. It is stored in the memory for a period defined in the item *History_Buffer_Removal*. One buffer occupies 30 bytes of memory. If archived data are stored by the DMS computer each 30 minutes then for 24 hours 48 buffers are needed.
Default value	- by default, 50000 buffers are used.

Parameter:	
number	- maximal number of buffers for historical data.

☑ ***Time_Difference=number***

Meaning	- the item is used to declare the time difference between clock indication of an **asix** system computer and a DMS computer. Even if the time displayed on the DMS computer is the same as the time of the **asix** system computer the time received by means of a serial interface, as a number of seconds from 01.01.1970, may be different from the actual time of the DMS computer.
Default value	- by default, this time is equal 3600 seconds.
Parameter:	
number	- time difference in seconds.

☑ ***REPORT=name***

Meaning	- the item allows to declare a place where the report read from a DMS computer will be sent by appropriate setting of pseudo-variable *REPORT*. As the purpose place of the report: - printer name, - disk file name may be entered.
Default value	- by default, the report transferred from a DMS computer is sent to a printer (LPT1).
Parameter:	
name	- printer name or disk file name.

☑ ***Max_history=number***

Meaning	- the item allows to declare a period of time counted from the current moment backwards, for which historical data in DMS computer memory will be read.
Default value	- by default, it is a period of 35 days.
Parameter:	
number	- time in days.

EXAMPLE 1

In the example a DMS computer named SIERSZA is defined. It is connected to the COM2 port. If the difference between the **asix** system computer time and the DMS computer time will exceed 5 seconds, then a clock synchronization occurs.

```
[ASMEN]
.....
SIERSZA=DMS500,COM2
....

[SIERSZA]
Auto_Sync=50
```

EXAMPLE 2

In the example the DMS computers with the following names are defined:
SIERSZA1,
SIERSZA2,
SIERSZA3,
SIERSZA4,
SIERSZA5,
SIERSZA6.

The DMS computers named as follows are connected to the COM2 port of the **asix** system computer:
SIERSZA1,
SIERSZA2,
SIERSZA3.

The DMS computers named as follows are connected to the COM3 port of the **asix** system computer:
SIERSZA4,
SIERSZA5.

The DMS computer named SIERSZA6 is connected to the COM4 port.

The COM2 and COM4 ports work with a speed of 19200 baud. The COM3 port works with a speed of 9600 baud.

The clocks of all DMS computers (except SIERSZA6) are synchronized with the clock of the **asix** system computer when the time difference exceeds 60 seconds. The clock of the DMS computer SIERSZA6 is not synchronized.

Report read by means of the pseudo-variable REPORT are printed on the printer LPT1: An exception is the DMS computer SIERSZA6, the reports of which are stored in the file *C:\RAP\SIERSZA6.RAP*.

```
[ASMEN].....SIERSZA1=DMS500,COM2SIERSZA2=DMS500,COM2
SIERSZA3=DMS500,COM2
SIERSZA4=DMS500,COM3
SIERSZA5=DMS500,COM3
SIERSZA6=DMS500,COM4
....
[DMS500];Default values for all DMS computers
DMSbaud=19200Auto_Sync=60 [DMS500:3];Default value for DMS
computers connected to the COM3 port
baud=9600

[SIERSZA6]
Auto_Sync=0
REPORT=C:\RAP\SIERSZA6.RAP
```

1.20. DSC - Driver of DSC PLC Protocol

Driver Use

The DSC driver is used for data exchange between an **asix** system computer and the DSC 2000 PLC. The data exchange is performed by means of standard serial interfaces of the **asix** system computer.

The cooperation of the **asix** system with the DSC 2000 PLC does not require any controller's program adaptation.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel operating according to the DSC protocol is given below:

logical_name=DSC,*id,port*

where:

id	- number assigned to the DSC PLC,
port	- name of the serial port e.g. COM1.

EXAMPLE

An example declaration of transmission channel based on the DSC Protocol.

CHAN1=DSC,5,COM3

The logical channel named CHAN1 has the following parameters defined:
- DSC protocol,
- data exchange is performed with the controller no. 5,
- COM3 serial port is used for data exchange.

Addressing the Process Variables

The syntax of symbolic address which is used for variables belonging to the DSC driver channel is as follows:

I<index>

where:

I	- symbol of the variable type, the same for all process variables of the DSC protocol;
index	- **number in hexadecimal format** identifying the process variable. Legal numbers are only those specified

in the *Nummer* item in *"Beschreibung der Rechnerschnittstelle"*, page 3.

All the variables except variables I10 and I11 (status of alarms) are variables the values of which may be read and written. The status of alarms may be read only.

Values of all process variables of the DSC 2000 controller are sent to the **asix** system in the form of 16-bit fix-point unsigned number. This principle is valid for floating-point and fixed-point variables. For that reason in order to show a floating-point value of process variable it is necessary to convert the value received from the controller to the floating-point format by means of a conversion function (most often ANALOG_FP).

EXAMPLE

Examples of declaration of process variables:

```
# set value of chlorine - floating-point number, two decimal places
X1,   I17,     CHAN1, 1,  1, ANALOG_FP,0,1000,0.0,10.0

# set value of pH - floating-point number, two decimal places
X2,   I2A,     CHAN1, 1,  1, ANALOG_FP,0,1000,0.0,10.0

# alarms and flags 1 - 16-bit fixed-point number
X3,   I10,     CHAN1, 1,  1, NOTHING

# alarms and flags 2 - 16-bit fixed-point number
X4,   I11,     CHAN1, 1,  1, NOTHING

# access code to the operator panel - 16-bit fixed-point number
X5,   I15,     CHAN1, 1,  1, NOTHING
```

The DSC driver is loaded as a DLL automatically.

1.21. DXF351 - Driver of Compart XF351 Device Protocol

Driver Use

The DXF351 driver is used for data exchange between Compart DXF351 devices of Endress+Hauser and an **asix** system computer. The communication is performed by using serial interfaces in the RS232C standard.

Compart DXF351 must be set to the following mode:

RS 2323 USAGE	- PRINTER
DEVICE ID	- any
BAUD RATE	- 9600
PARITY	- NONE
HANDSHAKE	- NONE

Settings in PRINT LIST:

ERRORS	- NO
ALARMS	- NO

The other items	- freely chosen:
PRINT INTERVAL	- 00:01 (data transfer every 1 minute)

Declaration of Transmission Channel

The full syntax of declaration of transmission channel operating according to the DXF351 protocol is given below:

logical_channel_name=DXF351, port

where:

DXF351	- protocol name,
port	- port name: COM1, COM2 etc.

EXAMPLE

The logical channel CHAN1 declaration working according to the DXF351 protocol on the COM2 port is as follows:

```
CHAN1=DXF351, COM2
```

The DXF351 driver is loaded as a DLL automatically.

Addressing the Process Variables

The syntax of symbolic address which is used for variables belonging to the DXF351 driver channel is as follows:

P<index>

where:

index	- number of measurement on the list PRINT LIST (see the table below).

Table 14. PRINT LIST.

P1	HEAT FLOW
P2	HEAT TOTAL
P3	HEAT GRAND TOTAL
P4	MASS FLOW
P5	MASS TOTAL
P6	MASS GRAND TOTAL
P7	COR. VOLUME FLOW
P8	COR. VOLUME TOTAL
P9	COR. VOL. GRAND TOTAL
P10	VOLUME FLOW
P11	VOLUME TOTAL
P12	VOL. GRAND TOTAL
P13	TEMPERATURE 1
P14	TEMPERATURE 2
P15	DELTA TEMPERATURE
P16	PROCESS PRESSURE
P17	DENSITY
P18	SPEC. ENTHALPY
P19	VISCOSITY
P20	REYNOLDS NUMBER

DXF351 transmits only these parameters, which are enclosed to the list **PRINT LIST** during configuring the Compart DXF351 device (group COMMUNICATION).

Raw values of all process variables are of FLOAT type.

An example of variable declarations:

```
X1,  Mass Flow ,           P4,    CHAN1,  1,  1,  NOTHING_FP
X2,  Volume Flow,          P10,   CHAN1,  1,  1,  NOTHING_FP
X3,  Temperature 1,        P13,   CHAN1,  1,  1,  NOTHING_FP
```

Driver Configuration

The DXF351 driver may be configured by using the **[DXF351]** section placed in the application INI file. Particular parameters are included in separate items of the section. Each item has the following syntax:

item_name=[number [,number]] [YES|NO]

☑ ***LOG_FILE=file_name***

Meaning	- the item allows to define a file where all diagnostic messages of the DXF351 driver and information about the content of telegrams received by the SPA driver will be written. If the item does not define the full path, then the log file is created in the current directory. The log file should be used only while the **asix** start-up.
Default value	- by default, the log file is not created.

EXAMPLE

LOG_FILE=D:\ASIX\DXF.LOG

☑ ***CHAR_TIMEOUT=number***

Meaning	- the item allows to determine the maximal time, which may pass between successive characters of a data block from DXF351. After having exceeded this time the DXF351 driver assumes that message as finished and begins analyzing the message content.
Default value	- by default, the item is set to 600 (millisecond).
Parameter:	
number	- number in milliseconds.

☑ ***LIMIT_OF_ERRORS = number***

Meaning	- the item allows to define a number of successive transmission errors, after which an error status of measurement is set.
Default value	- by default, the item has a value of 3.
Parameter:	
number	- number of successive transmission errors, after which an error status is set.

1.22. CtEcoMUZ - Driver of ecoMUZ Protocol

Driver Use

The CtEcoMUZ driver is used for data exchange between the **asix** system and Microprocessor Protecting ecoMUZ Devices made by JM Tronik. The transmission is performed through serial links by means of serial interfaces in the RS-232 or RS-485 standard, depending on ecoMUZ's serial interface type.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel which operates according to the ecoMUZ protocol is given below:

Channel=UNIDRIVER, CtEcoMUZ, *Port=number*; *Nr=number [;Timeout=number]*

where:

UNIDRIVER	- name of the universal UNIDRIVER driver,
CtEcoMUZ	- name of the driver for communication with an ecoMUZ device,
Port	- number of the ecoMUZ's serial port,
Nr	- number of an ecoMUZ serviced by the channel,
Timeout	- max. timeout for the first character of the controller response (in milliseconds); it is passed 1000 milliseconds by default.

The following constant transmission parameters are passed:

- transmission speed - 9600 Bd,
- number of bits in a character - 8,
- parity check,
- number of stop bits - 1.

EXAMPLE

Exemplary declarations of channels for communication with the 1064 and 1125 ecoMUZs, by means of the COM2 serial port, and the 1068 ecoMUZ, by means of the COM1 serial port:

```
K1064 = UNIDRIVER, CtEcoMUZ, Port=2;Nr=1064
K1066 = UNIDRIVER, CtEcoMUZ, Port=2;Nr=1066
K1068 = UNIDRIVER, CtEcoMUZ, Port=1;Nr=1068
```

Addressing the Process Variables

The syntax of symbolic address which is used for variables belonging to the CtEcoMUZ driver channel is as follows:

<*type*>[.<*index*>]

where:

type	- variable type,
index	- index within the type – for the following types:

- measurements,
- settings of protections,
- device state.

Symbolic names of variable types (in parentheses the type of a raw variable value is given):

Read-only variable types:

P	-	measurements (FLOAT), index range 1- 4,
Z	-	settings of protections (FLOAT), index range 1 – 6,
S	-	device state (BYTE), index range 1 – 3.

Write-only variable types:

PP	-	activation of transmitters (WORD),
KS	**-**	signalling deletion (WORD),
ZI	-	protection test performance I>> i I>,
ZZ	-	performing the test of floating-short-circuit protection

Read-only virtual variable:

SK - communication status (WORD) (1 – o.k., 0 – lack or communication errors)

EXAMPLE

Examplary variable declarations – the K1064 channel services the 1064 ecoMUZ, the K1066 channel services the 1066 ecoMUZ:

```
JJ_10, value of current I1  MUZ1064,           P1,   K1064,  1,  1,  NOTHING_FP
JJ_20, value of current I3  MUZ1066,           P3,   K1066,  1,  1,  NOTHING _FP
JJ_30, setting of short-circuit unit current I>> MUZ1064, Z1,   K1064,  1,  1,  NOTHING _FP
JJ_40, setting of short-circuit unit time I>> MUZ1066,Z4,   K1066,  1,  1,  NOTHING _FP
JJ_50, state of device MUZ1066,                     S1,   K1066,  1,  1,  NOTHING _BYTE
JJ_60, configuration of device MUZ1064,             S2,   K1064  1,  1,  NOTHING _BYTE
JJ_70, activation of transmitter MUZ1064,           PP,   K1064, 1,  1,  NOTHING
JJ_80, deletion of signalling of MUZ1066,           KS,  K1066, 1,  1,  NOTHING
JJ_90, status of communication with MUZ1064,        SK,  K1064, 1,  1,  NOTHING
JJ_91, status of communication with MUZ1066,        SK,  K1066, 1,  1,  NOTHING
```

Driver Configuration

The driver is configured by using the separate **[CTECOMUZ]** section. Using this section, you may declare:

- log file,
- log file size,
- telegram log.

☑ ***LOG_FILE = log_file_name***

Meaning	- for diagnostic purposes the text-type log file, which messages about driver operation status are written into, is used.
Default value	- by default, the log file is not created.
Defining	- manual.

☑ ***LOG_FILE_SIZE=number***

Meaning	- this item is used to define the size of the log file defined with use of the LOG_FILE item.
Default value	- by default, the log file size is 10 MB.
Defining	- manual.

☑ ***LOG_OF_TELEGRAMS =YES | NO***

Meaning	- this item allows contents of telegrams transferred between the driver and controllers to be written into the log file (declared with use of the LOG_FILE item). The referred item should only be used in the asix system start-up stage.
Default value	- by default, value of this item is set to NO.
Defining	- manual.

Exemplary Driver Section

```
[CTECOMUZ]
LOG_FILE=d:\tmp\CtEcoMUZ\muz.log
LOG_FILE_SIZE =20
LOG_OF_TELEGRAMS =YES
```

1.23. FESTO - Driver of Diagnostic Interface for FESTO PLCs

Driver Use

The FESTO driver is used for data exchange with FESTO FST-103, FST-405, FST IPC PLCs by means of a diagnostic interface. Required version of firmware: 2.20 or later. The transmission is executed by means of serial interfaces in the V24 (RS232C) standard by using the standard serial ports of an **asix** system computer.

The operation of the **asix** system with FESTO PLCs by using a diagnostic interface does not require any controller's program adaptation.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel operating according to the FESTO PLC protocol is as follows:

logical_name=FESTO,*port,[baud,character,parity,stop,cpu_no]*

where:

FESTO	- driver name of the FESTO PLC diagnostic interface;
port	- name of the serial port;
baud	- transmission speed in baud;
character	- number of bits in a transmitted character;
parity	- parity check type;
stop	- number of stop bits,
cpu_no	- CPU number in the controller.

Parameters *baud, character, parity, stop, cpu_no* are optional. In case of omitting them the following default values are assumed:

- transmission speed - 9600 Bd,
- number of bits in a character - 8,
- parity check type - no parity check,
- number of stop bits - 1,
- CPU number - 0.

EXAMPLE

Example items declaring the use of two transmission channel working according to the protocol of FESTO controller diagnostic interface are given below. In both channels the communication is performed by means of the same physical interface but the data ara exchanged with other CPUs:

```
CHAN2=FESTO,COM1,9600,8,none,1,2,8
```

```
CHAN3=FESTO,COM1,9600,8,none,1,3,8
```

In the example above the declaration of channels differs only with the number of CPU. The CHAN2 channel allows for data exchange with the CPU numbered 2 whereas the CHAN3 channel with the CPU no. 3. The other parameters in the channel declarations are identical:

- port COM1,
- transmission speed of 9600 Bd,
- transmitted character length - 8 bits,
- no parity check,
- one stop bit.

Addressing the Process Variables

The syntax of symbolic address used for process variables supported by the FESTO driver:

VARIABLE_TYPE variable_index

where:

VARIABLE_TYPE	- string identifying the variable type,
variable_index	- variable index within the given type.

The following symbols of process variable types are allowable (range of variable indexes is specific for different types of controllers):

EW	- input words,
AW	- output words,
ESW	- words of input statuses,
ASW	- words of output statuses,
MW	- words of buffers,
TW	- current readings of timers,
TV	- set values of timers,
TA	- attributes of timers,
T	- readings of timers,
ZW	- current readings of counters ,
ZV	- set values of counters,
Z	- counter readings,
R	- registers.

EXAMPLES

R15	- register no. 15
EW0	- input word 0
AW8	- input output 8

All process variables are treated as 16-bit unsigned numbers.

The FESTO driver is loaded as a DDL automatically.

1.24. FILE2ASIX – Driver for Data Import from Files

Driver Use

FILE2ASIX is used for importing data into the **asix** system from text files of the following structure:

Opening line
Line containing variable 1 value
Line containing variable 2 value
...
Line containing variable n value
Closing Line

Each line has the following form:

P1<sep>P2<sep>P3<sep>P4<sep>P5

where:

<sep>	- delimiter ‘;’
P1	- for opening and closing line it is the file saving date and time in UTC format; - for other lines it is the variable reading time in UTC format;
P2	- symbolic address of the variable (it must not contain exclamation character ‘!’);
P3	- quality of the variable in one of forms given below: BAD , UNCERTAIN , GOOD;
P4	- value of the variable in integer or floating-point form (‘.’- as a delimiter)
P5	- name of the device path.

Fields: *P2*, *P3*, *P4* and *P5* have to remain empty for opening and closing line.

The UTC time format is as follows:

YYYY-MM-DD<SP>GG:NN:SS,MS

where:

YYYY	- 4 digits indicating a year,
MM	- 2 digits indicating a month,
DD	- 2 digits indicating a day,
SP	- white space character,
HH	- 2 digits indicating an hour,
NN	- 2 digits indicating minutes,
SS	- 2 digits indicating seconds,
MS	- 2 digits indicating milliseconds.

NOTICE *The file is treated as correct if first and last line are exactly the same.*

Declaration of Transmission Channel

The full syntax of declaration of transmission channel operating according to the FILE2ASIX has the following form:

channel_name=FILE2ASIX*, file_path [, period]*

where:

FILE2ASIX	- driver name;
file_path	- path to file containing variable values;
period	- time (given in seconds) between consecutive file reading (10 seconds are the default value).

EXAMPLE

# reading from the file *\\komp\c$\data\data.csv* with 5 second interval
CHANNEL1 = FILE2ASIX, \\KOMP\C$\Data\Data.csv, 5

# reading from the file *n:Data\data.csv* with 10 second inteval (default value)
CHANNEL2 = FILE2ASIX, n:Data\Data.csv

Addressing the Process Variables

The ASMEN process variable address may take one of the forms given below:

"access_path ! address"
"address"

where:

access_path	- generalized path to the device (value of the *P5* field),
address	- address of the variable in the path scope (value of the *P2* field).

The second form of addressing („*address*") is used when an access path is an empty string (*P5* field has no value).

Only a single variable may be passed through the file, thus a number of elements in the ASMEN variable declaration has to be equal to 1.

The driver automatically converts the type of the variable read from the file to a raw type, which is required by the conversion function given in the declaration of the ASMEN variable.

EXAMPLE

An example of the file which passes the values of ZMIENNA_FP and ZMIENNA_WORD variables:

```
2002-10-04 12:23:37,004;;;;
2002-10-04
12:23:26,999;ZMIENNA_FP;GOOD;32.4436;PLC:S7[BEL_SPREZ]
2002-10-04 12:23:26,999;ZMIENNA_WORD;GOOD;32;
2002-10-04 12:23:37,004;;;;
```

EXAMPLE

Examples of the ASMEN variable declaration for the above file:
JJ_1, "PLC:S7[BEL_SPREZ] ! ZMIENNA_FP", KANAL, 1, 1, NOTHING_FP
JJ_2, "ZMIENNA_WORD", KANAL, 1, 1, NOTHING

Driver Configuration

Driver parameters are set by using the separate **[FILE2ASIX]** section. The following entries may be placed in this section:

- log file creation,
- log file size,
- data validity period.

Names of entries connected with the log file are strictly refer to the convention used in other ASMEN drivers.

☑ ***LOG_FILE=file_name***

Meaning	- it allows definition of a file in which all diagnostic messages of the driver will be saved. If position does not define full path, then the log file will be created in the current directory. The log file should only be used at the stage of the **asix** system start-up.
Default value	- by default, no log file is created.

☑ ***LOG_FILE_SIZE=number***

Meaning	- it allows to specify the log file size (using MB unit).
Default value	- by default, 1MB size is assumed.
Parameter:	
number	- a size of the log file in MB.

☑ ***DATA_VALIDITY_PERIOD =number***

Meaning	- this entry is used for monitoring cases when a data file with a given interval reading fail. The entry defines the number of consecutive failed readings, after which status of the variables will be set to BAD.
Default value	- by default, a period of data validity is equal to two cycles of data reading from a joint computer.
Parameter:	
number	- a number of failed reading cycles, after which an error status is set.

1.25. FP1001 - Driver of METRONIC Measurer Protocol

Driver Use

The FP1001 driver is used for data exchange between FP1001 3.W or FP1001 3.4 flow monitors manufactured by METRONIC Kraków and an **asix** system computer provided with a serial interface in the RS485 standard.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel operating according to the FP1001 protocol is given below:

logical_name=FP1001, *id, type, COMi [,baud]*

where:

id	- device no. in the network;
type	- device type: 1 – steam flow monitor FP1001 3.4, 2 – water flow monitor FP1001 3.W;
COMi	- name of the serial port;
baud	- transmission speed (by default 9600).

The FP1001 driver is loaded as a DLL automatically.

Addressing the Process Variables

The way of addressing the process variables is given in the following tables.

Table 15. Addressing the FP1001 3.W Meter (Water Monitor).

Symb. Address	Variable in FP-1001	Conversion Type	Allowed Operation
	Values of basic measurements		
P1	Instantaneous value Fm	Byte7->Float	Reading
P2	Instantaneous value Q	Byte7->Float	Reading
P3	Instantaneous value p	Byte4->Float	Reading
P4	Instantaneous value T	Byte3->Float	Reading
P5	Instantaneous value Tw	Byte3->Float	Reading
P6	Calculation value u water	Byte6->Float	Reading
P7	Adder state Fm	Byte10->Float	Reading
P8	Adder state Q	Byte10->Float	Reading
	Alarms and markers		
F1	Marker of unit "M" or "G"	Byte->Word	Reading
F2	Marker of emergency value substitution for T "A" or " "	Byte->Word	Reading
F3	Marker of emergency value substitution for "A" or " "	Byte->Word	Reading
F4	State of alarms for Fm : "X", "L" or "H"	Byte->Word	Reading
F5	State of alarms for Q : "X", "L" or "H"	Byte->Word	Reading
F6	State of alarms for p : "X", "L" or "H"	Byte->Word	Reading
F7	State of alarms for T : "X", "L" or "H"	Byte->Word	Reading
	Programmed discrete parameters		
ND1	Input type F (0-20mA, 4-20mA)	Byte->Word	Reading
ND2	Input type p (0-20mA, 4-20mA)	Byte->Word	Reading
ND3	Characteristic	Byte->Word	Reading
ND4	Measurement unit (kg/h and MJ/h or t/h and GJ/h)	Byte->Word	Reading
ND5	Measurement of pressure p (absolute or manometer))	Byte->Word	Reading
ND6	Output 4-20mA (Fm or Q)	Byte->Word	Reading
ND7	Impulse output (Fm or Q)	Byte->Word	Reading
ND8	Alarm Fm (H) (off, on)	Byte->Word	Reading
ND9	Alarm Fm (L) (off, on)	Byte->Word	Reading
ND10	Alarm Q (H) (off, on)	Byte->Word	Reading
ND11	Alarm Q (L) (off, on)	Byte->Word	Reading
ND12	Alarm p (H) (off, on)	Byte->Word	Reading
ND13	Alarm p (L) (off, on)	Byte->Word	Reading
ND14	Alarm T (H) (off, on)	Byte->Word	Reading
ND15	Alarm T (L) (off, on)	Byte->Word	Reading

Table 16. Addressing the FP1001 3.W Meter (Water Monitor) (continuation).

Symb. Address	**Variable in FP-1001**	**Conversion Type**	**Allowed Operation**
	Programmed numeric parameters		
NL1	Range F max	Byte5->Float	Reading
NL2	Cut-off level Fc	Byte5->Float	Reading
NL3	Range p max	Byte4->Float	Reading
NL4	Emergency temperature TA	Byte3->Float	Reading
NL5	Emergency water temperature TwA	Byte3->Float	Reading
NL6	Design temperature To	Byte3->Float	Reading
NL7	Output current range	Byte5->Float	Reading
NL8	Alarm level Fm (H)	Byte5->Float	Reading
NL9	Alarm level Fm (L)	Byte5->Float	Reading
NL10	Alarm level Q (H)	Byte5->Float	Reading
NL11	Alarm level Q (L)	Byte5->Float	Reading
NL12	Alarm level p (H)	Byte4->Float	Reading
NL13	Alarm level p (L)	Byte4->Float	Reading
NL14	Alarm level T (H)	Byte3->Float	Reading
NL15	Alarm level T (L)	Byte3->Float	Reading
	Working times		
T1	Working time of device	Byte5+2->Dword	Reading
	Conversion of letter and digital markers on bits		
	letter "A"	bit 0	
	letter "G"	bit 1	
	letter "H"	bit 2	
	letter "L"	bit 3	
	letter "M"	bit 4	
	letter "N"	bit 5	
	letter "S"	bit 6	
	letter "X"	bit 8	
	space	bit 9	
	digit "0"	bit 0	
	digit "1"	bit 1	

Table 17. Addressing the FP1001 3.4 Meter (Steam Monitor).

Symb. Address	Variable in FP-1001	Conversion Type	Allowed Operation
	Values of basic measurements		
P1	Instantaneous value Fm	Byte7->Float	Reading
P2	Instantaneous value Fw	Byte6->Float	Reading
P3	Instantaneous value Q	Byte7->Float	Reading
P4	Instantaneous value Qw	Byte7->Float	Reading
P5	Instantaneous value deltaQ	Byte7->Float	Reading
P6	Instantaneous value p	Byte4->Float	Reading
P7	Instantaneous value T	Byte3->Float	Reading
P8	Calculation value p	Byte6->Float	Reading
P9	Calculation value u	Byte6->Float	Reading
P10	Instantaneous value Tw	Byte3->Float	Reading
P11	Calculation value u water	Byte6->Float	Reading
P12	Adder state Fm	Byte10->Float	Reading
P13	Adder state Fw	Byte10->Float	Reading
P14	Adder state Q	Byte10->Float	Reading
P15	Adder state Qw	Byte10->Float	Reading
P16	Adder state delta Q	Byte10->Float	Reading
	Alarms and markers		
F1	Marker of unit "M" or "G"	Byte->Word	Reading
F2	Marker of emergency value substitution for p "A" or " "	Byte->Word	Reading
F3	Marker of emergency value substitution for T "A" or " "	Byte->Word	Reading
F4	Marker of steam type "N", "S" or "A"	Byte->Word	Reading
F5	State of alarms for Fm : "X", "L" or "H"	Byte->Word	Reading
F6	State of alarms for Q : "X", "L" or "H"	Byte->Word	Reading
F7	State of alarms for p : "X", "L" or "H"	Byte->Word	Reading
F8	State of alarms for T : "X", "L" or "H"	Byte->Word	Reading

Table 18. Addressing the FP1001 3.4 Meter (Steam Monitor) (continuation).

Symb. Address	Variable in FP-1001	Conversion Type	Allowed Operation
	Values of conditional adders		
S1	Fm for saturated steam	Byte10->Float	Reading
S2	Fm for >Fm (H)	Byte10->Float	Reading
S3	Fm for <Fm (L)	Byte10->Float	Reading
S4	Fm for >Q (H)	Byte10->Float	Reading
S5	Fm for <Q (L)	Byte10->Float	Reading
S6	Fm for >p (H)	Byte10->Float	Reading
S7	Fm for <p (L)	Byte10->Float	Reading
S8	Fm for >T (H)	Byte10->Float	Reading
S9	Fm for <T (L)	Byte10->Float	Reading
S10	Q for saturated steam	Byte10->Float	Reading
S11	Q for >Fm (H)	Byte10->Float	Reading
S12	Q for <Fm (L)	Byte10->Float	Reading
S13	Q for >Q (H)	Byte10->Float	Reading
S14	Q for <Q (L)	Byte10->Float	Reading
S15	Q for >p (H)	Byte10->Float	Reading
S16	Q for <p (L)	Byte10->Float	Reading
S17	Q for >T (H)	Byte10->Float	Reading
S18	Q for <T (L)	Byte10->Float	Reading
	Programmed discrete parameters		
ND1	Steam type (dry or saturated)	Byte->Word	Reading
ND2	Measurement (p or T)	Byte->Word	Reading
ND3	Measuring method (reducer or Vortex)	Byte->Word	Reading
ND4	Return water (lack, only Tw, Tw and Fw)	Byte->Word	Reading
ND5	Input type F (0-20mA, 4-20mA,0-10kHz,01kHz)	Byte->Word	Reading
ND6	Input type Fw (0-20mA, 4-20mA,0-10kHz,01kHz)	Byte->Word	Reading
ND7	Input type p (0-20mA, 4-20mA,0-10kHz,01kHz)	Byte->Word	Reading
ND8	Input type T (0-20mA, 4-20mA,Pt100)	Byte->Word	Reading
ND9	Characteristic F	Byte->Word	Reading
ND10	Measurement unit (kg/h and MJ/h or t/h and GJ/h)	Byte->Word	Reading
ND11	Pressure measurement p (absolute or manometer.)	Byte->Word	Reading
ND12	Output 4-20mA (Fm or delta Q)	Byte->Word	Reading
ND13	Impulse output (Fm or delta Q)	Byte->Word	Reading
ND14	Alarm Fm (H) (off, on)	Byte->Word	Reading
ND15	Alarm Fm (L) (off, on)	Byte->Word	Reading
ND16	Alarm Q (H) (off, on)	Byte->Word	Reading
ND17	Alarm Q (L) (off, on)	Byte->Word	Reading
ND18	Alarm p (H) (off, on)	Byte->Word	Reading
ND19	Alarm p (L) (off, on)	Byte->Word	Reading
ND20	Alarm T (H) (off, on)	Byte->Word	Reading
ND21	Alarm T (L) (off, on)	Byte->Word	Reading

Table 19. Addressing the FP1001 3.4 Meter (Steam Monitor) (continuation).

Symb. Address	Variable in FP-1001	Conversion Type	Allowed Operation
	Programmed Discrete Parameters		
ND22	Conditional adder Fm for saturated steam (off, on)	Byte->Word	Reading
ND23	Conditional adder Fm for >Fm (H) (off, on)	Byte->Word	Reading
ND24	Conditional adder Fm for <Fm (L) (off, on)	Byte->Word	Reading
ND25	Conditional adder Fm for >Q (H) (off, on)	Byte->Word	Reading
ND26	Conditional adder Fm for <Q (L) (off, on)	Byte->Word	Reading
ND27	Conditional adder Fm for >p (H) (off, on)	Byte->Word	Reading
ND28	Conditional adder Fm for <p (L) (off, on)	Byte->Word	Reading
ND29	Conditional adder Fm for >T (H) (off, on)	Byte->Word	Reading
ND30	Conditional adder Fm for <T (L) (off, on)	Byte->Word	Reading
ND31	Conditional adder Q for saturated steam (off, on)	Byte->Word	Reading
ND32	Conditional adder Q for >Fm (H) (off, on)	Byte->Word	Reading
ND33	Conditional adder Q for <Fm (L) (off, on)	Byte->Word	Reading
ND34	Conditional adder Q for >Q (H) (off, on)	Byte->Word	Reading
ND35	Conditional adder Q for <Q (L) (off, on)	Byte->Word	Reading
ND36	Conditional adder Q for >p (H) (off, on)	Byte->Word	Reading
ND37	Conditional adder Q for <p (L) (off, on)	Byte->Word	Reading
ND38	Conditional adder Q for >T (H) (off, on)	Byte->Word	Reading
ND39	Conditional adder Q for <T (L) (off, on)	Byte->Word	Reading
	Programmed Numeric Parameters		
NL1	Range F max	Byte5->Float	Reading
NL2	Cut-off level Fc	Byte5->Float	Reading
NL3	Range Fw max	Byte5->Float	Reading
NL4	Cut-off level for Fw	Byte5->Float	Reading
NL5	Range p max	Byte4->Float	Reading
NL6	Emergency pressure pA	Byte4->Float	Reading
NL7	Design pressure	Byte4->Float	Reading
NL8	Range T min	Byte3->Float	Reading
NL9	Range T max	Byte3->Float	Reading
NL10	Emergency temperature TA	Byte3->Float	Reading
NL11	Design temperature	Byte3->Float	Reading
NL12	Emergency water temperature	Byte3->Float	Reading
NL13	Output current range	Byte5->Float	Reading
NL14	Alarm level Fm (H)	Byte5->Float	Reading
NL15	Alarm level Fm (L)	Byte5->Float	Reading
NL16	Alarm level Q (H)	Byte5->Float	Reading
NL17	Alarm level Q (L)	Byte5->Float	Reading
NL18	Alarm level p (H)	Byte4->Float	Reading
NL19	Alarm level p (L)	Byte4->Float	Reading
NL20	Alarm level T (H)	Byte3->Float	Reading
NL21	Alarm level T (L)	Byte3->Float	Reading

Driver Configuration

The FP1001 protocol driver may be configured by using the **[FP1001]** section, which is placed in the **asix** application INI file. Individual parameters are transferred in separate items of the section. Each item has the following syntax:

item_name=[number [,number]] [YES] [NO]

☑ ***LOG_FILE=file_name***

Meaning	- the item allows to define a file to which all the diagnostic messages of the FP1001 driver and the information about the content of the telegrams received and sent by the FP1001 driver will be written. If the item doesn't define its full path, the log file will be created in the current directory.
Default value	- by default, the file log isn't created.

☑ ***LOG_OF_TELEGRAMS=YES|NO***

Meaning	- the item allows to write to the log file the content of telegrams transmitted between **asix** and FP1001 devices. Writing the telegrams content to the log, file should be used only during the **asix** start-up.
Default value	- by default, the FP1001 driver does not write the content of telegrams to the log file.

☑ ***TRANSMISSION_DELAY = number***

Meaning	- the item is used to declare a pause between transmissions. The pause is expressed as a number of 10 ms intervals.
Default value	- by default, the item is set to 1 (10 msec).

☑ ***UPDATE=number***

Meaning	- the item defines a number of seconds, after which the driver updates the contents of its internal buffers by reading the measure data from FP1001 devices FP1001.
Default value	- by default, the item is set to 1 (updating every 1 second).

☑ ***NUMBER_OF_REPETITIONS=number***

Meaning	- the item allows to define a number of repetitions in case of occurring the transmission error.
Default value	- by default, the item is set to 0 (no repetitions).

1.26. GFCAN - Driver of CANBUS Protocol for CanCard

Driver Use

The GFCAN driver is used for data exchange between devices with the CAN network interface and an **asix** system computer provided with a CAN network communication processor card manufactured by Garz & Fricke Industrieautomation GmbH and provided with the Garz & Fricke's CAN driver for Windows NT" version 1.0.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel operating according to the GFCAN protocol is given below:

logical_name=GFCAN

The GFCAN driver is loaded as a DLL automatically.

Addressing the Process Variables

Values of process variables are transferred in telegrams sent by controllers connected to the CAN network. Each telegram consists maximally of 8 bytes, which may be identified as:

bytes with indexes 1 – 8	(type BY),
16-bit numbers with indexes 1 – 4	(type WD),
32-bit numbers with indexes 1 – 2	(type DW),
32-bit floating-point numbers with indexes 1 – 2	(type FP).

The GFCAN driver differs the following access types to process variables:

read only	(type R_),
write only	(type W_),
write/read	(type RW_).

The addressing the process variables consists in the indication of:

- access type (R_, W_ or RW_);
- variable type (BY, WD, DW, FP);
- telegram no. (for variables with RW_ access type it is the telegram number that is used to read the variable);

- index within the telegram (for variables with RW_ access type it is the index in the telegram that is used to read the variable);
- for variables with RW_ access type it is necessary to declare additionally:
  - a/ telegram number that is used to read the variable,
  - b/ index in the telegram that is used to write the variable.

The syntax of symbolic address which is used for variables belonging to the GFCAN driver channel is as follows:

<access_type><variable_type><tel>.<index>[.<tel>.<index>]

where:

access type	- access type to a process variable:
R_	- only reading,
W_	- only writing,
RW_	- reading and writing,
variable_type	- process variable type:
BY	- variable of the byte type,
WB	- variable of the 16-bit number type,
DW	- variable of the 32-bit number type,
FP	- variable of the 32-bit floating-point number type.
tel	- telegram number,
index	- index within the telegram.

EXAMPLE

```
X1, bytes 1-4 of telegram 31,R_FP31.2,          NONE, 1, 1, NOTHING_FP
X2, word no. 3 of telegram 31,R_WD31.3,         NONE, 1, 1, NOTHING
X3, state of burners,         RW_BY31.1.35.3,   NONE, 1, 1, NOTHING_BYTE
X4, valve setting,            RW_WD32.1.34.1,   NONE, 1, 1, NOTHING
```

The variable X1 is a variable of the 32-bit floating-point number type transferred to the **asix** system in bytes 1,2,3 and 4 of the telegram no. 31.

The variable X2 is a variable of the 16-bit number type, the value of which is transferred to the **asix** system by bytes 5 and 6 (third word) of telegram no.31. The value of the variable can't be modified by the application (variable only to read).

Value of the variable X3 is transferred to the **asix** system by means of the byte no. 1 of the telegram no. 31. The value exchange of the variable X3 consists in sending from the **asix** system the telegram no. 35, the byte no. 3 of which includes the required value of the variable X3.

Driver Configuration

The driver of GFCAN protocol may be configured by using the **[GFCAN]** section, placed in the application INI file. Individual parameters are transferred in separate items of the section. Each item has the following syntax:

item_name=[number [,number]] [YES] [NO]

☑ ***TRANSMISSION_SPEED=baud_id***

Meaning	- the item is used to declare a transmission speed in the CAN network.
Default value	- by default, the item is set to 500 kB.
Parameter:	
baud_id	- identifier of transmission speed in the CAN network:

1000	-	1 MB
800	-	800 kB
500	-	500 kB
250	-	250 kB
125	-	125 kB
100	-	100 kB
50	-	50 kB
20	-	20 kB
10	-	10 kB

EXAMPLE

An exemplary declaration of the transmission speed of 125 kB:

```
TRANSMISSION_SPEED=125
```

☑ ***NETWORK_CONTROL=number***

Meaning	- the item allows to test the reception of telegrams from the CAN network. It defines the maximal time between receptions of successive telegrams with the same number. In case of exceeding this time, the process variables bound with such telegram will be provided with an error status. If additionally in the same time any telegram was not received from the CAN network, a message about a lack of telegrams in network is generated in *'Control Panel'*.
Default value	- by default, the GFCAN driver does not check reception of telegrams.
Parameter:	
number	- maximal number of seconds, which may pass between successive telegrams with the same number.

EXAMPLE

An exemplary declaration of checking reception of telegrams every 5 seconds:

```
NETWORK_CONTROL=5
```

☑ ***TELEGRAM_TRACE=YES|NO***

Meaning	- the item controls transferring to the operator panel the messages about telegrams that have been received from the CAN network. A message includes the number of telegram, number of bytes and content of individual telegrams in hexadecimal form.
Default value	- by default, the contents of telegrams are not displayed.

EXAMPLE

An exemplary declaration of tracing the content of received telegrams:

```
TELEGRAM_TRACE=YES
```

☑ ***CONTROL_TRACE=YES|NO***

Meaning	- the item controls transferring to the operator panel the messages about control telegrams that have been sent from the **asix** system computer to controllers. A message includes the number of control telegram, number of bytes and telegram contents in hexadecimal form.
Default value	- by default, the contents of telegrams are not displayed.

EXAMPLE

An example of tracing the control telegrams:

```
CONTROL_TRACE=YES
```

☑ ***LOG_FILE=file_name***

Meaning	- the item allows to define a file to which all diagnostic messages of GFCAN driver and information about content of telegrams received and sent by the GFCAN driver will be written. If the item does not define a full path, the log file will be created in the current directory. The log file should be used only while the **asix** system start-up.
Default value	- by default, the contents of telegrams are not displayed.

☑ ***MAX_MOTOROLA_TEL=number***

Meaning	- the item allows to specify a maximal number of telegram, the content of which will be converted according to MOTOROLA format. All the telegrams with numbers, which are bigger than declared by means of the item MAX_MOTOROLA_TEL, will be converted according to INTEL format.
Default value	- by default it is assumed that all telegrams are converted according to INTEL format.

EXAMPLE

An example of declaration as a result of which the telegrams with numbers up to 150 inclusive are converted according to MOTOROLA format:

```
MAX_MOTOROLA_TEL=150
```

1.27. K3N - Driver of OMRON's K3N Meters Family Protocol

Driver Use

The K3N driver is used for data exchange between the OMRON's K3N meters family and **asix** system computers. The communication is executed according to the CompoWay/F protocol from OMRON via standard serial ports of an **asix** computer.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel which uses the K3N protocol is given below:

logical_channel_name=K3N, *id, port [, baud, character, parity, stop]*

where:

K3N	- driver name;
id	- controller number in the K3N network;
port	- port name: COM1, COM2 etc.;
optional parameters:	
baud	- transmission speed;
character	- number of bits in a character;
parity	- parity check type;
stop	- number of stop bits.

If the optional parameters are not given, then the factory settings of meter are assumed as default values, i.e.:

- transmission speed: 9600 baud,
- bits in a character: 7,
- parity check: EVEN,
- number of stop bits: 2.

EXAMPLE

The declaration of the logic channel named CHAN1, which operates according to the K3N protocol and exchanges data with the meter numbered 1 through the COM2 port with default transmission parameters:

CHAN1 = K3N, 1, COM2

The K3N driver is loaded as a DLL automatically.

Addressing the Process Variables

The syntax of symbolic address which is used for variables belonging to the K3N driver channel is as follows:

<type>[.<category>.<index>]

where:

type	- variable type;
category	- category within the type (for some types);
index	- index within the category (for some types).

Symbols of variable types (in parentheses the type of raw variable value is given):

VW	- values of variables transferred in the form of 16-bit unsigned numbers (WORD). It concerns to Status data variable (category C0, index 3);
VL	- values of variables transferred in the form of 32-bit signed numbers (LONG). It concerns the variables of the category C0 except the variable Status data;
PW	- values of parameters transferred in the form of 16-bit unsigned numbers (WORD). It concerns to all the variables of the category 8000 and 8824;
PL	- values of parameters transferred in the form of 32-bit signed numbers (LONG). It concerns to all the variables of the category C00C and 8824;
STAT	- status of work mode (WORD);
INFO	- supplemental information transferred with the status of work mode (WORD);
RST	- execution of the command *reset of minimal and maximal values* (WORD);
FCLR	- execution of the command *clear forced-zero setting* (WORD);
MODE	- setting the operating mode of the meter (*Run* or *Setting Mode*) (WORD);
CTRL	- setting the programming mode of the meter (*Remote programming* or *Local programming*) (WORD).

Category and *index* must be given for types **VW**, **VL**, **PW**, **PL**. The list of legal symbols (numbers) of category and index ranges within individual category contains the documentation *"Communication Output-type Intelligent Signal Processor – OPERATION MANUAL" Cat. No. N96-E1-1 (pkt 1-5 'Memory/Parameters Area Details')*.

Values of **VW**, **VL**, **PW** or **PL** type variables may be read and modified. A correct reading of the **VW**, **VL**, **PW**, **PL** type variables is possible only when the meter is set in the *Run mode.*

Values of **INFO**, **STAT** type variables may be only read.

The **RST**, **FCLR**, **MODE**, **CTRL** type variables are used only to execute commands controlling the operating mode of the meter and it is not possible to read them.

Sending the control by using a **MODE** type variable causes:

- switching the meter to the *Run mode* if the control value of the variable is set on 0,
- switching the meter to *Setting mode* if the control value of the variable is different from 0.

Sending the control by using a **CTRL** type variable causes:

- switching the meter to the *Local programming* mode if the control value of the variable is set on 0,
- switching the meter to the *Remote programming* mode if the control value is different from 0.

EXAMPLES OF VARIABLE DECLARATIONS

```
# Present value: variable no. 0 from the category C0
JJ_0,  VL.C0.0,   CHAN1, 1, 1, NOTHING_LONG

# Maximum value: variable no. 1 from the category C0
JJ_1,  VL.C0.1,   CHAN1, 1, 1, NOTHING_LONG

# status value (Status data): variable no 3 from the category C0
JJ_2,  VW.C0.3,   CHAN1, 1, 1, NOTHING

# current operating mode of the meter (Run, Setting mode)
JJ_10, STAT,      CHAN1, 1, 1, NOTHING

# supplement information transferred with the meter status
# (HOLD status, RESET status, Local/Remote programming)
JJ_11, INFO,      CHAN1, 1, 1, NOTHING

# execution of the command: reset of minimal and maximal values of the
meter
JJ_12, RST,       CHAN1, 1, 1, NOTHING

# execution of the command: Clear forced-zero setting
JJ_13, FCLR,      CHAN1, 1, 1, NOTHING

# execution of the command: Switch mode
JJ_14, MODE,      CHAN1, 1, 1, NOTHING

# execution of the command: Remote/Local programming
JJ_14, CTRL,      CHAN1, 1, 1, NOTHING

# parameter input range of the K3NX meter
JJ_20,  PW.8000.0, CHAN1, 1, 1, NOTHING

# parameter scaling display value 2 of the K3NX meter
JJ_21,  PL.C00C.1, CHAN1, 1, 1, NOTHING_LONG

# parameter power supply frequency of the K3NX meter
JJ_22,  PW.8824.0, CHAN1, 1, 1, NOTHING
```

```
# parameter input mode of the K3NC meter
JJ_30,  PW.8000.0, CHAN1, 1, 1, NOTHING

# parameter power failure memory of the K3NC meter
JJ_31,  PW.8824.1, CHAN1, 1, 1, NOTHING

# parameter compensation value of the K3NC meter
JJ_32,  PW.C82A.0, CHAN1, 1, 1, NOTHING_LONG
```

Driver Configuration

The K3N protocol driver may be configured by means of the **[K3N]** section placed in the application INI file. Individual parameters are transferred in separate items of the section. Each item has the following syntax:

item_name=[number [,number]] [YES] [NO]

☑ ***LOG_FILE=file_name***

Meaning	- the item allows to define a file where all diagnostic messages of the K3N driver and information about the contents of telegrams received by the driver will be written. If the item does not define its full path, then the log file is created in the current directory. The log file should be used only while the **asix** start-up.
Default value	- by default, any log file is not created.

☑ ***LOG_FILE_SIZE=number***

Meaning	- the item allows to define the log file size in MB.
Default value	- by default, it is assumed that the log file has a size of 1 MB.

☑ ***LOG_OF_TELEGRAMS=YES|NO***

Meaning	- the item allows to write to a log file (declared by using the item LOG_FILE) the contents of telegrams sent within the communication with the meter. Writing the contents of telegrams to the log file should be used only while the **asix** start-up.
Default value	- by default, the telegrams are not written.

☑ ***RECV_TIMEOUT=id,number***

Meaning	- the item allows to determine a maximal waiting time for arriving the first character of the answer from a given meter. After overflow of this time it is assumed that the considered meter is turned off and the transmission session ends with an error**.**
Default value	- by default, it is assumed that the maximal waiting time for the first character of the answer is equal to 1000 milliseconds.
Parameter:	
id	- meter no. in the K3N network,
number	- time in milliseconds (from 100 up to 5000).

☑ ***CHAR_TIMEOUT=id,number***

Meaning	- the item allows to determine a maximal time between successive characters of the answer from a given meter. After having exceeded this time it is assumed that the considered meter does not work correctly and the transmission session ends with an error.
Default value	- by default, it is assumed that the maximal time between successive characters of the answer is equal 50 milliseconds.
Parameter:	
id	- number of the meter in the network,
number	- time in milliseconds (from 10 up to 300).

☑ ***NUMBER_OF_REPETITIONS=number***

Meaning	- the item allows to determine a number of repetitions in case of occurring the transmission error.
Default value	- by default, the item receives a value of 0 (no repetition).

1.28. K-BUS - Driver of Protocol for VIESSMANN Decamatic Boiler PLCs

Driver Use

The K-BUS driver is used for data exchange between VIESSMANN Dekamatic boiler controllers connected to the Dekatel-G (Vitocom 200) concentrator and an **asix** system computer. For communication with **asix** an interface of the RS-232C standard is used.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel which uses the K-BUS protocol is given below:

logical_channel_name=K-BUS, *id*, *port [, alarm_offset]*

where:

K-BUS	- driver name,
id	- identifier of the regulator,
port	- port name: COM1, COM2 etc.,
optional parameters:	
alarm_offset	- offset added to the number of alarm sent from the regulator. By default, the value of offset equals 0.

The list of identifiers assigned to individual regulators is given below:

1	-	Dekamatic-D1/Dekamatic-DE
2	-	Dekamatic-D2 (Kesselregelung 2. Kessel)
3	-	Dekamatic-D2 (Kesselregelung 3. Kessel)
4	-	Dekamatic-HK (1. und 2. Heizkreis)
5	-	Dekamatic-HK (3. und 4. Heizkreis)
6	-	Dekamatic-HK (5. und 6. Heizkreis)
7	-	Dekamatic-HK (7. und 8. Heizkreis)
8	-	Dekamatic-HK (9. und 10. Heizkreis)
9	-	Dekamatic-HK (11. und 12. Heizkreis)
10	-	Dekamatic-HK (13. und 14. Heizkreis)

11 - Dekamatic-HK (15. und 16. Heizkreis)

Transmission parameters are constant:
- 1200 Bd,
- 8 bits of a character,
- parity check: even,
- one stop bit.

EXAMPLE

A declaration of the logical channel named CHAN1, which works according to the K-BUS protocol and exchanges the data with the regulator Dekamatic-DE (id 1) through the COM2 port is as follows:

```
CHAN1 = K-BUS, 1, COM2
```

The K-BUS driver is loaded as a DLL automatically.

Addressing the Process Variables

The syntax of symbolic address which is used for variables belonging to the K-BUS driver channel is as follows:

V<index>

where:

V	- fixed symbol of the variable type,
index	- variable index, compatible to the table of addresses of variables for the controller under consideration (given in HEX form).

Raw values of the variables are transferred by the driver as numbers of WORD type.

EXAMPLES

Examples of declaration of variables:

```
#  max boiler temperature (id 12 HEX)
JJ_1,  V12,  CHAN1,  1,  1,  NOTHING

#  external temperature (id 25 HEX )
JJ_2,  V25,  CHAN1,  1,  1,  NOTHING
```

Driver Configuration

The K-BUS protocol driver may be configured by using the **[K-BUS]** section placed in the application INI file. Individual parameters are transferred in separate items of the section. Each item has the following syntax:

item_name=[number [,number]] [YES] [NO]

☑ ***LOG_FILE=file_name***

Meaning	- the item allows to define a file where all diagnostic messages of the K-BUS driver and information about the content of telegrams received by the driver are written. If the item does not define its full path, then the log file is created in the current directory. The log file should be used only while the **asix** start-up.
Default value	- by default, the log file is not created.

☑ ***LOG_OF_TELEGRAMS=YES|NO***

Meaning	- the item allows to write to the log file (declared by using the item LOG_FILE) the contents of telegrams sent within the communication with regulators. Writing the contents of telegrams to the log file should be used only while the **asix** start-up.
Default value	- by default, the telegrams are not written.

☑ ***LOG_FILE_SIZE=number***

Meaning	- the item allows to define the log file size in MB.
Default value	- by default, it is assumed that the log file has a size of 1 MB.

☑ ***RECV_TIMEOUT=id,number***

Meaning	- the item allows to determine a maximal waiting time for arriving the first character of the answer from a given regulator. After this time is over it is assumed that the controller under consideration is turned off and the transmission session ends with an error.
Default value	- by default, it is assumed that the maximal waiting for the first character of the answer is equal to 1000 milliseconds.
Parameter:	
id	- number of the regulator,
number	- time in milliseconds (from 100 up to 5000).

☑ ***CHAR_TIMEOUT=id,number***

Meaning	- the item allows to determine a maximal time between successive characters of the answer from a given regulator. After having exceeded this time it is assumed that the regulator under consideration does not work correctly and the transmission session ends with an error.
Default value	- by default, it is assumed that the maximal time between successive characters of the answer is equal to 50 milliseconds.
Parameter:	
id	- number of the regulator,
number	- time in milliseconds (from 10 up to 300).

Delay After Having Mapped the Data in Concentrator Dekatel-G (Vitocom 200)

Dekatel-G (Vitocom 200) concentrators allow simultaneous reading 8 variables maximally. The work mode with the concentrator consists in successive execution of the following functions for the successive group of variables:

- transfer of a list of maximally 8 variables to the concentrator (so called concentrator mapping),
- wait for updating the variables in the concentrator after mapping,
- reading the variables from the concentrator.

☑ ***MAPPING_DELAY=number***

Meaning	- the item allows to determine the time, which must elapse between mapping and the first reading of data from the concentrator so that the read data might be assumed as reliable. In case of too short delay the risk of reading the values of variables which were registered in the concentrator before mapping exists.
Default value	- by default, the parameter assumes a value of 35 seconds.
Parameter:	
number	- time in seconds.

NOTE *The protocol specification does not give the formula to calculate the delay after concentrator mapping, therefore this parameter must be determined experimentally by the user.*

☑ ***GLOBAL_ALARMS=YES|NO***

Meaning	- the item controls the way of transferring alarms read from regulators to the alarms system of **asix** start-up.
Default value	- by default, the alarms are transferred to the alarms system as global alarms (transferred to the alarms system by means of the function *AsixAddAlarmGlobalMili()*). Setting the value of the item GLOBAL_ALARMS on NO causes that the alarms are transferred to the alarms system by means of the function *AsixAddAlarmMili()*.

☑ ***SIGNED_VARIABLE = YES/NO***

Meaning	- the item determines the way of interpretation of the BYTE variable. Setting the value on NO makes the UNSIGNED CHAR interpretation be given to the variable.
Default value	- by default, and in the case of setting the variable on YES the variable of the BYTE type assumes the SIGNED CHAR interpretation – it allows transferring the negatives.

1.29. CtLG - Driver of Dedicated Protocol of LG Master-K and Glofa GM PLCs

Driver Use

The CtLG driver is designed to exchange data between the **asix** system and LG Industrial Systems Master – K and Glofa GM PLCs with use of an RS232 port. The driver enables access to LG PLC data addressed directly by passing the address of the variable within the device. The driver allows simultaneous handling of many LG PLCs.

Declaration of Transmission Channel

The syntax of declaration of transmission channel using the driver is as follows:

Channel=UNIDRIVER, CtLG, *Port=port_number*; *Speed=transmission_speed*; *StopBits=number_of_stop_bits*;*ParityBit=control_of_frame_parity*; *DataBits=number_of_data_bits*; *NrOfBlocks=number_of_blocks*; *TimeSynchr=address[:period]*;

where:

UNIDRIVER	- name of universal UNIDRIVER;
CtLG	- name of driver used for communication with LG PLC;
Port	- number of the COM serial port;
Speed	- speed of transmission between computer and device; the following speeds are acceptable: 1200, 2400, 4800, 9600, 19200, 38400, 56000, 57600, 115200, 128000; every speed is given in bits per second, i.e. bauds; a default value is 1200 Bd;
StopBits	- number of stop bits: 1 or 2; for the GlofaGM6 PLC this parameter is built into the controller on a permanent basis and amounts to 1; a default value of the parameter is 1;
ParityBit	- defines the method of frame parity control; available options: *no*, *parity_control*, *odd_parity_contol*; as substitutes for these options you may use, respectively: 0, 1, 2; for GlofaGM6 PLC this parameter is built into the controller on a permanent basis and amounts to 0, i.e. no; a default value of the parameter is *parity_control*;
DataBits	- number of data bits per frame: 7 or 8; for the GlofaGM6 PLC this parameter is built into the controller on a

	permanent basis and amounts to 8; a default value of the parameter is 8;
NrOfBlocks	- max number of blocks specifying variables in a single read operation; a maximum value of the parameter is 16; the parameter has been introduced because GLOFA PLC properly executes queries for max 4 blocks (inconsistently with specification); a default value of the parameter is 16;
TimeSynchr	- for Glofa PLCs it is a direct address of the table of 9 bytes, in the controller that PC's system time is to be entered into. The table will be filled in with BCD-code numbers as follows:

time[0] = two younger digits of year
time[1] = month
time[2] = day of month
time[3] = hour
time[4] = minutes
time[5] = seconds
time[6] = day of week (Monday - 0, Tuesday – 1,... Sunday - 6)
time[7] = two older digits of year
time[8] = 1

e.g. 2001 – 03 – 15 18:30:45 Tuesday: time[0] = 01, time[1] = 03, time[2] = 15, time[3] = 18, time[4] = 30, time[5] = 45, time[6] = 03, time[7] = 20,

On Glofa controller's side, you should execute the command *RTC_SET argument.* The argument is a symbolic address of the table of 8 bytes. When declaring this variable, in the *Memory allocation* field select the *Assign(AT)* option and pass the direct address of the variable in the transmission channel declaration. Upon the system time is rewritten from the PC to the controller, the value 1 is assigned to the ninth element of the table.

In case of the MASTER-K PLC, *TimeSynchr* is the address of the table of word type consisting of five elements, which is available in the controller N. The table is filled in with BCD-code numbers as follows:

time[0] = older byte = 2 younger digits of year, younger byte = month (1..12)
time[1] = older byte = day of month (1..31), younger byte = hour
time[2] = older byte = minutes, younger byte = seconds
time[3] = older byte = 2 older digits of year, younger byte = day of week (Sunday – 0, Monday – 1...Saturday – 6)
time[4] = 1

With the exception of *time[4]*, these words should be rewritten into a special area of the memory and the appropriate bit should be set.

period	- default parameter to define the interval in seconds at which time will be rewritten from the PC to the controller. By default, the synchronization time is 60 seconds.

EXAMPLE

An example of declaration of channel in which time will be synchronised for the controller numbered 6 (GLOFA type) by the write into the area starting with MB10 (every 25 seconds):

PLC1 = UNIDRIVER, CtLG, Port=2;Speed=9600; StopBits=1; ParityBit=odd_parity_control; DataBits=8; TimeSynchr=6.MB10:25

Addressing the Process Variables

Addressing the Variables in Master – K family

The following direct address is only acceptable:

ControllerNo.TypeOfDevice.Address

where:

Controller no - number between 0 and 31.
Type of device - *(see: the table below).*

Table 20. Types of Devices in Master-K Family.

Type of Device	Range of Device	Read / Write	Bit / Word
P (Input/Output relay)	P0 ~ P0031 (32 words) P0.0 ~ P31.15 (32 × 16 bits)	Read/Write	Both
M (auxiliary relay)	M0 ~ M191 (192 words) M0.0 ~ M191.15 (192 × 16 bits)	Read/Write	Both
K (keep relay)	K0 ~ K31 (32 words) K0.0 ~ K31.15 (32 × 16 bits)	Read/Write	Both
L (link relay)	L0 ~ L63 (64 words) L0.0 ~ L63.15 (64 × 16 bits)	Read/Write	Both
F (special relay)	F0 ~ F63 (64 words) F0.0 ~ F63.15 (64 × 16 bits)	Read	Both
T (timer contact relay)	T0.0 ~ T0.255 (256 bits)	Read/Write	Both
T (timer elapsed value)	T0 ~ T255 (256 words)	Read/Write	Both
C (counter contact relay)	C0.0 ~ C0.255 (256 bits)	Read/Write	Both
C (counter elapsed value)	C0 ~ C255 (256 words)	Read/Write	Both
S (step controller)	S0 ~ S99 (100 sets)	Read/Write	Word only
D (data register)	D0 ~ D4999 (5000 words)	Read/Write	Word only

NOTE *T and C devices should not be used for bit addressing because it does not work due to error in the controller's operating system.*

There are two types of variables:

- bit,
- word.

The variable address may contain up to 8 characters (without device type character and '.' character, if any). The address is given in decimal format. When bit is addressed within a word, the bit number (from 0 to 15) is given after a dot. An exception to this rule is Timer and Counter types. As you can see in the above table, bits for these types are addressed from 0 to 255.

For example, 0. M5 – word with the address 5
0. M5.10 – eleventh bit in the word with the address 5

EXAMPLE

Examples of variable declarations:

JJ_00, variable of WORD type with address M1, 0.M1, PLC1, 1, 1, NIC
JJ_01, variable of BIT type with address M5.10, 0.M5.10, PLC1, 1, 1, NIC

Addressing the Variables in Glofa – GM Family

The direct address has the following format:

ControllerNo.TypeOfDevice.TypeOfVariable.Address

where:

ControllerNo	- defines the controller number and is a number between 0 and 31;
TypeOfDevice	- defines the device type; the following types are available:

- M (internal memory),
- Q (output),
- I (Input).

The range of addressing for these devices is configurable and depends on the type of device.

All of these devices can be both written to and read from.

TypeOfVariable - defines the variable type. The following types are available:

- X – bit,
- B – byte
- W – word,
- D – double word.

For the M device the address is given in decimal format and may contain maximum 13 characters, without the character of the device and variable type, e.g.:

3.MW1 - word with the address 1 from the dictionary no 3

If you want to address a bit in the M device within a byte, word or double word, the bit number (counted from 0) in decimal format should be given after a dot, for example:

4.MW1.14 - fourteenth bit in the word 1 from the dictionary 4
5.MD2.30- thirteenth bit in the double word 2 from the dictionary 5

Bit can also be addressed directly using the X character, e.g.:

0.MX10 - tenth bit.

In case of addressing Q and I devices, the address is given in decimal format. These are three numbers (*base*, *slot, number*) separated with the '.' character, e.g.:

2.QX3.1.4 - controller no 2, 3 base, 1 slot, 4th bit,
3.IW2.4.1 - controller no 3, 2 base, 4 slot, 1st word.

EXAMPLE

Examples of variable declarations:

JJ_00, VARIABLE OF WORD TYPE WITH ADDRESS MW1, 0.MW1, PLC1, 1, 1, NIC
JJ_01, variable of BIT type with address QX3.1.4, 0.QX3.1.4, PLC1, 1, 1, NIC

Time Marker

Values of variables read from LG are assigned a PC's time stamp.

Driver Parameterization

The driver parameterization takes place with use of the separate section named **[CTLG]** which is placed in the initialisation file of an **asix** application. Using this section, you may declare:

- log file,
- log file size,
- telegram log.

☑ ***LOG_FILE = log_file_name***

Meaning	- for diagnostic purposes the text-type log file into which messages about driver operation status are written is used.
Default value	- by default, the log file is not created.
Defining	- manual.

☑ ***LOG_FILE_SIZE=number***

Meaning	- this item is used to define the size of the log file defined with use of the LOG_FILE item.
Default value	- by default, the log file size is 1 MB.
Defining	- manual.

☑ ***LOG_OF_TELEGRAMS =yes | No***

Meaning	- this item allows contents of telegrams transferred between the driver and controllers to be written into the log file (declared with use of the LOG_FILE item). The referred item should only be used in the **asix** system start-up stage.
Default value	- by default, value of this item is set to NO.
Defining	- manual.

EXAMPLE

An example of the driver section:

```
[CTLG]
LOG_FILE=d:\tmp\ctLG\LG.log
LOG_FILE_SIZE=3
```

1.30. LUMBUS - Driver for LUMEL Meters

Driver Use

The LUMBUS driver is used for data exchange between RG72 controllers manufactured by Lubuskie Zakłady Aparatów Elektrycznych (Electrical Measuring Instrument Works) "LUMEL" in Zielona Góra and an **asix** system computer. The communication is executed by means of serial interfaces in the RS485 standard.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel operating according to the LUMBUS protocol is given below:

logical_channel_name=LUMBUS, *number, port, baud*

where:

LUMBUS	- driver name,
number	- controller no. in the network,
port	- port name: COM1, COM2 etc.,
baud	- transmission speed in the range 1200 – 9600 Bd.

By default it is assumed:
- transmission speed 9600 Bd,
- number of character bits - 8,
- without parity check (PARITY NONE),
- number of stop bits - 1.

EXAMPLE

The declaration of the logical channel named CHANNEL, which works according to the LUMBUS driver protocol and exchanges data with the RG72 regulator numbered 1 through the COM2 port with a speed of 4800 Bd, is as follows:

```
CHANNEL=LUMBUS, 1, COM2, 4800
```

The LUMBUS driver is loaded as a DLL automatically.

Addressing the Process Variables

The syntax of symbolic address which is used for variables belonging to the LUMBUS driver channel is as follows:

<*type*><*index*>[.*subindex*]

where:

type	- variable type; allowed types: P - single measurement, PT - table of measurements, WT - array of free days, DT - array of holiday's dates;
index	- according to the specification given in point 3 of the *"Serial interface RS-485 in RG7-07/2 controller"* documentation; for single measures *index* takes the index value assigned to the measurement in the table; for the values transferred in the form of arrays *index* takes the index value assigned to the array, and the item of the variable under consideration in the array is specified by *subindex*;
subindex	- applies to the specification of variables transferred in the form of arrays and determines the variable location in the array; the subindex of the first element in the array takes a value of 0.

The raw value of measure is of FLOAT type.
The raw value of free day and holiday date is an ASCII string in dd.mm.yyyy format ended by zero (including 11 character).

EXAMPLES

Examples of declarations of variables the values of which are transferred individually:

```
X13,  hour of turning on night reduction, P13,        CHANNEL, 1, 1,  NOTHING_FP
X23,  set temperature c.w.u,                P23,        CHANNEL, 1, 1,  NOTHING_FP
```

Examples of declarations of variables, the values of which are transferred in form of arrays:

```
X39,  set temperature in control room,      PT38.0,  CHANNEL, 1, 1,  NOTHING_FP
X40,  ext. temp (A) – initial point of curve,  PT38.1,  CHANNEL, 1, 1,  NOTHING_FP
X50,  max. allowed temperature of return,  PT48.1,  CHANNEL, 1, 1,  NOTHING_FP
X56,  dead band of c.h.,                   PT52.3,  CHANNEL, 1, 1,  NOTHING_FP

X68,  economies – holidays day 1,         WT67.0,  CHANNEL, 11, 1, NOTHING_TEXT
X69,  economies – holidays day 2,         WT67.1,  CHANNEL, 11, 1, NOTHING_TEXT
X70,  economies – holidays day 3,         WT67.2,  CHANNEL, 11, 1, NOTHING_TEXT

X119, first period of vacation – from,      DT119.0, CHANNEL, 11, 1,NOTHING_TEXT
X120, first period of vacation – from,      DT119.1, CHANNEL, 11, 1,NOTHING_TEXT
X121, second period of vacation – from, DT121.0, CHANNEL, 11, 1,NOTHING_TEXT
X122, second period of vacation – from, DT121.1, CHANNEL, 11, 1,NOTHING_TEXT
X123, third period of vacation – from,    DT123.0, CHANNEL, 11, 1,NOTHING_TEXT
X124, third period of vacation – from,    DT123.1, CHANNEL, 11, 1,NOTHING_TEXT
```

Driver Configuration

The LUMBUS protocol driver may be configured by means of the **[LUMBUS]** section placed in the application INI file. Individual parameters are transferred in separated items of the section. Each item has the following syntax:

item_name=[number [,number]] [YES] [NO]

☑ ***LOG_FILE=file_name***

Meaning	- the item allows to define a file where all diagnostic messages of the LUMBUS driver and the information about the contents of telegrams received by the driver are written. If the item does not define the full path, then the LOG_FILE is created in the current directory. The log file should be used only while the **asix** start-up.
Default value	- by default, the log file is not created.

EXAMPLE

LOG_FILE=D:\asix\LUMBUS.LOG

☑ ***LOG_OF_TELEGRAMS=YES|NO***

Meaning	-the item allows to write to the log file (declared by using the item LOG_FILE) the contents of telegrams sent within the communication with the RG72 regulator. Writing the contents of telegrams to the log file should be used only while the **asix** start-up.
Default value	- by default, telegrams are not written.

☑ ***NUMBER_OF_REPETITIONS=number***

Meaning	- the item allows to determine a number of repetitions in case of an appearance of transmission error.
Default value	- by default, the item assumes a value of 0 (no repetitions).

Table 21. List of Symbolic Addresses.

Symb. Address	Index	Designation of Measurements From RG72	Conversion Type	Allowed Operation
P9	9		Word->Float	R
P10	10	f.active	Byte->Float	RW
P11	11	Tpwrt	Word->Float	RW
P12	12	DeltaT	Float->Float	RW
P13	13	night hours	Char->Float	RW
P14	14	day hours	Char->Float	RW
P15	15	free days	Byte->Float	RW
P16	16	temp. of summer	Float->Float	RW
P17	17	l. Days	Char->Float	RW
P18	18	hours Pom	Char->Float	RW
P19	19	Pump	Byte->Float	RW
P20	20	cw_oszcz	Byte->Float	RW
P21	21	Tzew	Byte->Float	RW
P22	22	St.pompa	Byte->Float	RW
P23	23	T.zad.cw	Float->Float	RW
P24	24	Priority	Byte->Float	RW
P25	25	t.prio	Byte->Float	RW
P26	26	t_pwrcwu	Byte->Float	RW
P27	27	t_progcwu	Byte->Float	RW
P28	28	Disinfection	Byte->Float	RW
P29	29	clock mode	Byte->Float	RW
PT31.0	31	Lkan	Char->Float	R
PT31.1	31	selection of curve, Sommer_is	Char->Float	R
P36	36	lock_full	Bit->Float	W
P37	37	lock_part	Bit->Float	W
		co – heating function		
PT38.0	39	T.zad.pk	Float->Float	RW
PT38.1	40	T.zew(A)	Float->Float	RW
PT38.2	41	T.co(A)	Float->Float	RW
PT38.3	42	tg.alfa	Float->Float	RW
PT38.4	43	T.zew(B)	Float->Float	RW
PT38.5	44	tg.beta	Float->Float	RW
PT38.6	45	T.co_max	Float->Float	RW
PT38.7	46	delta_co	Float->Float	RW
P47	47	T.freeze	Float->Float	RW
		co (centr. heat.) – return curve		

Table 22. List of Symbolic Addresses (continuation).

Symb. Address	Index	Designation of Measurements from RG72	Conversion Type	Allowed Operation
PT48.0	49	T.pwr_min	Float->Float	RW
PT48.1	50	T.pwr_max	Float->Float	RW
PT48.2	51	tg (pwrt)	Float->Float	RW
		Pid		
PT52.0	53	xp co	Int->Float	RW
PT52.1	54	ti co	Int->Float	RW
PT52.2	55	td co	Int->Float	RW
PT52.3	56	2N co	Int->Float	RW
PT52.4	57	H co	Int->Float	RW
PT52.5	58	to co	Int->Float	RW
PT52.6	59	tp co	Int->Float	RW
PT52.7	60	xp cw	Int->Float	RW
PT52.8	61	ti cw	Int->Float	RW
PT52.9	62	td cw	Int->Float	RW
PT52.10	63	2N cw	Int->Float	RW
PT52.11	64	H cw	Int->Float	RW
PT52.12	65	to cw	Int->Float	RW
PT52.13	66	tp cw	Int->Float	RW
		Holidays and free days		
WT67.0	68	holidays/free no. 1	Word->ASCII(11)	RW
WT67.1	69	holidays/free no. 2	Word->ASCII(11)	RW
WT67.2	70	holidays/free no. 3	Word->ASCII(11)	RW
WT67.3	71	holidays/free no. 4	Word->ASCII(11)	RW
WT67.4	72	holidays/free no. 5	Word->ASCII(11)	RW
WT67.5	73	holidays/free no. 6	Word->ASCII(11)	RW
WT67.6	74	holidays/free no. 7	Word->ASCII(11)	RW
WT67.7	75	holidays/free no. 8	Word->ASCII(11)	RW
WT67.8	76	holidays/free no. 9	Word->ASCII(11)	RW
WT67.9	77	holidays/free no. 10	Word->ASCII(11)	RW
WT67.10	78	holidays/free no. 11	Word->ASCII(11)	RW
WT67.11	79	holidays/free no. 12	Word->ASCII(11)	RW
WT67.12	80	holidays/free no. 13	Word->ASCII(11)	RW
WT67.13	81	holidays/free no. 14	Word->ASCII(11)	RW
WT67.14	82	holidays/free no. 15	Word->ASCII(11)	RW
WT67.15	83	holidays/free no. 16	Word->ASCII(11)	RW
WT67.16	84	holidays/free no. 17	Word->ASCII(11)	RW
WT67.17	85	holidays/free no. 18	Word->ASCII(11)	RW
WT67.18	86	holidays/free no. 19	Word->ASCII(11)	RW
WT67.19	87	holidays/free no. 20	Word->ASCII(11)	RW

Table 23. List of Symbolic Addresses (continuation).

Symb. Address	Index	Designation of Measurements from RG72	Conversion Type	Allowed Operation
WT67.20	88	holidays/free no. 21	Word->ASCII(11)	RW
WT67.21	89	holidays/free no. 22	Word->ASCII(11)	RW
WT67.22	90	holidays/free no. 23	Word->ASCII(11)	RW
WT67.23	91	holidays/free no. 24	Word->ASCII(11)	RW
WT67.24	92	holidays/free no. 25	Word->ASCII(11)	RW
WT67.25	93	holidays/free no. 26	Word->ASCII(11)	RW
WT67.26	94	holidays/free no. 27	Word->ASCII(11)	RW
WT67.27	95	holidays/free no. 28	Word->ASCII(11)	RW
WT67.28	96	holidays/free no. 29	Word->ASCII(11)	RW
WT67.29	97	holidays/free no. 30	Word->ASCII(11)	RW
WT67.30	98	holidays/free no. 31	Word->ASCII(11)	RW
WT67.31	99	holidays/free no. 32	Word->ASCII(11)	RW
WT67.32	100	holidays/free no. 33	Word->ASCII(11)	RW
WT67.33	101	holidays/free no. 34	Word->ASCII(11)	RW
WT67.34	102	holidays/free no. 35	Word->ASCII(11)	RW
WT67.35	103	holidays/free no. 36	Word->ASCII(11)	RW
WT67.36	104	holidays/free no. 37	Word->ASCII(11)	RW
WT67.37	105	holidays/free no. 38	Word->ASCII(11)	RW
WT67.38	106	holidays/free no. 39	Word->ASCII(11)	RW
WT67.39	107	holidays/free no. 40	Word->ASCII(11)	RW
WT67.40	108	holidays/free no. 41	Word->ASCII(11)	RW
WT67.41	109	holidays/free no. 42	Word->ASCII(11)	RW
WT67.42	110	holidays/free no. 43	Word->ASCII(11)	RW
WT67.43	111	holidays/free no. 44	Word->ASCII(11)	RW
WT67.44	112	holidays/free no. 45	Word->ASCII(11)	RW
WT67.45	113	holidays/free no. 46	Word->ASCII(11)	RW
WT67.46	114	holidays/free no. 47	Word->ASCII(11)	RW
WT67.47	115	holidays/free no. 48	Word->ASCII(11)	RW
WT67.48	116	holidays/free no. 49	Word->ASCII(11)	RW
WT67.49	117	holidays/free no. 50	Word->ASCII(11)	RW
		Holidays and free days		
DT119.0	119	first period of vacation (from)	Int->ASCII(11)	RW
DT119.1	120	first period of vacation (from)	Int->ASCII(11)	RW
DT121.0	121	second period of vacation (from)	Int->ASCII(11)	RW
DT121.1	122	second period of vacation (from)	Int->ASCII(11)	RW
DT123.0	123	third period of vacation (from)	Int->ASCII(11)	RW
DT123.1	124	third period of vacation (from)	Int->ASCII(11)	RW
DT125.0	125	forth period of vacation (from)	Int->ASCII(11)	RW
DT125.1	126	forth period of vacation (from)	Int->ASCII(11)	RW
DT127.0	127	fifth period of vacation (from)	Int->ASCII(11)	RW
DT127.1	128	fifth period of vacation (from)	Int->ASCII(11)	RW
		presence or lack of sensors		

Table 24. List of Symbolic Addresses (continuation).

Symb. Address	Index	Designation of Measurements from RG72	Conversion Type	Allowed Operation
PT129.0	130	sensor 1	Byte->Float	RW
PT129.1	131	sensor 2	Byte->Float	RW
PT129.2	132	sensor 3	Byte->Float	RW
PT129.3	133	sensor 4	Byte->Float	RW
PT129.4	134	sensor 5	Byte->Float	RW
PT129.5	135	sensor 6	Byte->Float	RW
		temperature differences for sensors		
PT136.0	137	sensor 1	Byte->Float	RW
PT136.1	138	sensor 2	Byte->Float	RW
PT136.2	139	sensor 3	Byte->Float	RW
PT136.3	140	sensor 4	Byte->Float	RW
PT136.4	141	sensor 5	Byte->Float	RW
PT136.5	142	sensor 6	Byte->Float	RW
		time of full valve opening		
PT143.0	144		Byte->Float	RW
PT143.1	145		Byte->Float	RW
		start/stop		
PT146.0	147		Byte->Float	RW
PT146.1	148		Byte->Float	RW
		safety codes		
PT149.0	150		Word->Float	RW
PT149.1	151		Word->Float	RW
PT149.2	152		Word->Float	RW
		current time		
PT153.0	154	Year	Byte->Float	RW
PT153.1	155	Month	Byte->Float	RW
PT153.2	156	Day	Byte->Float	RW
PT153.3	157	Hour	Byte->Float	RW
PT153.4	158	Minute	Byte->Float	RW
		sensor error		
PT173.0	174	error of sensor 1	Char->Float	R
PT173.1	175	error of sensor 2	Char->Float	R
PT173.2	176	error of sensor 3	Char->Float	R
PT173.3	177	error of sensor 4	Char->Float	R
PT173.4	178	error of sensor 5	Char->Float	R
PT173.5	179	error of sensor 6	Char->Float	R
		control signal		
PT180.0	181		Float->Float	R
PT180.1	182		Float->Float	R

Table 25. List of Symbolic Addresses (continuation).

Symb. Address	Index	Designation of Measurements from RG72	Conversion Type	Allowed Operation
		measured temperatures		
PT183.0	184	temperature 1	Float->Float	R
PT183.1	185	temperature 2	Float->Float	R
PT183.2	186	temperature 3	Float->Float	R
PT183.3	187	temperature 4	Float->Float	R
PT183.4	188	temperature 5	Float->Float	R
PT183.5	189	temperature 6	Float->Float	R
		alarm data		
PT240.0	241	alarm 1	Float->Float	R
PT240.1	242	alarm 2	Float->Float	R
PT240.2	243	alarm 3	Float->Float	R
		remote control of valve of centr. heat. (co) and heat water (cw)		
PT247.0	248		Char->Float	RW
PT247.1	249		Char->Float	RW
		remote control of pump of co and cw		
PT250.0	251		Char->Float	RW
PT250.1	252		Char->Float	RW
		switching on/off the pump of co and cw		
PT253.0	254		Char->Float	W
PT253.1	255		Char->Float	W
		opening the valve of co and cw		
PT256.0	257		Char->Float	W
PT256.1	258		Char->Float	W
		closing the valve of co, cw		
PT259.0	260		Char->Float	W
PT259.1	261		Char->Float	W
		factory settings		
P262	262		Byte->Float	W
P263	263	fact. settings Co	Byte->Float	W
P264	264	fact. settings Cwu	Byte->Float	W
P265	265	fact. settings Others	Byte->Float	W

1.31. MACMAT - Driver of GAZ_MODEM Protocol for MACMAT Station

Driver Use

The MACMAT driver is used for communication with the MACMAT station. The driver supports the stations signed as Korektor Impulsowy (Impulse Corrector) 01723 CMK 01 97/01/02 manufactured by COMMON Ltd. and PKNMiJ 03-03-93 RP T-Zw5-1 manufactured by the PLUM company.

Declaration of Transmission Channel

The logical channel is defined by placing an appropriate item in the [ASMEN] section of the application INI file. The syntax of declaration of transmission channel operating according to the COMLI protocol is given below:

logical_name=MACMAT,*address*,*COMn*

where:

n	- number of the serial port to which the network of MACMAT stations is connected;
address	- station address.

Driver Configuration

Each defined channel may have its own section, name of which is its logical name i.e. *[logical_name]*. The COMn port may also have its own section named **[MACMAT:n]**. Values defined in such section become default values for all stations connected to a given port. If in the INI file a section named **[MACMAT]** is placed, then the values placed in such section become default values for all stations supported by the controller. Values placed in the section of a given station (*[logical_name]*) have a priority before values placed in the section of a given serial port and the last ones have a priority before values placed in the [MACMAT] section. If the parameter is not present in any section, then its default value is taken according to the description below. In particular, the initialization file may not include any sections parameterizing stations. Appropriate records are required only in the [ASMEN] section.

Parameters of transmission with use of a serial interface cannot be placed in sections concerning individual stations.

☑ ***Auto_sync=number***

Meaning	- if the parameter is different from 0, then automatic synchronization of the computer clock with the MACMAT station clock. The parameter value determines the minimal time between sequent comparisons of station and computer clocks. A comparison of clocks is performed only during other data reading from the station.
Default value	- 3600 (1 hour).
Parameter:	
number	- time in seconds.

☑ ***Alt_port = COMm, metod_of_switching_to_alternative_port, metod_of_switching_to_basic_port***

Meaning	- *(See: Definition of alternative ports)*.

AsComm =Yes/No

Meaning	- if yes is given, then the driver will use AsComm Connection Manager to establish connections with MACMAT stations.
Default value	- No.

☑ ***No_Errors =Yes/No***

Meaning	- if yes is given, then the driver will not output the messages on errors in lines and timeout.
Default value	- No.

☑ ***Max_History_Buffers=number***

Meaning	- determines the maximal number of buffers containing historical data, read for needs of the archiving module. One buffer includes historical data from one time interval for one variable. It is stored in the memory within the time specified by the parameter *Max_History_Buffers*. One buffer occupies about 400 bytes of memory and can contain 50 values. If archive data are saved by the station every 15 minutes thus, for 24 hours, 2 buffers for one variable are necessary. Historical buffers are used by the data archiving program ASPAD during completing a B type archive. After the time determined by the parameter *History_Buffer_Removal* buffers are removed from the memory.
Default value	- 5000.
Parameter:	
number	- number of buffers.

☑ ***Max_history =number***

Meaning	- determines a time period in days, counted from the current moment backwards, for which historical data, stored in the station memory, will be read.
Default value	- 35.
Parameter:	
number	- time in days.

☑ ***Max_Time_Difference =number***

Meaning	- the maximal time difference between indications of a station clock and a computer clock, after exceeding of which synchronization of clocks occurs. The parameter has a meaning only when the *Auto_sync* parameter is different from zero.
Default value	- 60.
Parameter:	
number	- time in seconds.

☑ ***Status_Mask =number***

Meaning	- a number determining which values of the variable status cause non-validity of the variable value. The variable status is read from the MACMAT station together with its value. This status is a bitmap; meaning of bits is described in the station documentation. The driver executes the logical operation AND on the variable status received from the controller and on the value of the parameter *Status_Mask*. If the result of this operation is different from zero, then the data value is invalid. The data value is invalid too if the variable status has a value of 0 (i.e. lack of data).
Default value	- the default, value of 6 means that values, exceeding the measuring range, are non-validated. As the parameter value, an integer, whose individual bits correspond to appropriate bits of the status, should be given.
Parameter:	
number	- time in seconds.

☑ ***Counter_Ratio =number***

Meaning - gas flow counter is transmitted in the form of two floating-point numbers *Vn0* and *Vn1*.

Protocol specification states that the counter value is calculated according to the formula:

Vn0 + Vn1*10000

However, some stations use the formula:

Vn0 + Vn1*100000

Parameter defines the value, by which the quantity *Vn1* should be multiplied:

Vn0 + Vn1*Counter_Ratio.

Default value	- 10000.

☑ ***P_Ratio =number***

Meaning	- according to the MacMAT station specification, the pressure is expressed in kPa. However, some stations transfer the pressure expressed in MPa. It concerns only archive (register) data. The parameter determines, by what factor the pressure, transferred by the station, should be multiplied.
Default value	- 1000.

☑ ***baud =number***

☑ ***bps=number***

Meaning	- transmission speed.
Default value	- 9600.
Parameter:	
number	- value passed in Bd.

☑ ***parity =parity_parameter***

Meaning	- item determines a parity type.
Default value	- n.
Parameter:	
number	- parity type: n – no parity but, o – odd parity check, e – even parity check, m – mark, s – space.

☑ ***retries =number***

☑ ***retry=number***

Meaning	- number of transmission repetitions in case of transmission errors.
Default value	- 5.

☑ ***word =number***

☑ ***word_length =number***

Meaning	- word length.
Default value	- 8.
Parameter:	
number	- number from the range from 5 to 8 bits.

☑ ***time-out =number***

☑ ***timeout =number***

Meaning	- waiting time for station answer in seconds.
Default value	- 2.

☑ ***History_Buffer_Removal =number***

Meaning	- parameter determines time after which buffers containing historical data, read for needs of the archiving module, are removed.
Default value	- 30.
Parameter:	
number	- the time is given in minutes.

☑ ***AllErrors =yes/no***

Meaning	- if the parameter has a value of no, then the information on *timeout* errors will appear in *'Control Panel'* only when the transmission missed in spite of attempts of its repetition. If it has a value of yes, then the information on all errors is transmitted to *'Control Panel'*.
Default value	- no.

☑ ***RTS =yes/no***

Meaning	- if yes is given, then data sending to the station will occur by means of the RTS line set on high state and the reception on low state.
Default value	- no.

☑ ***RTS_OFF_Delay =yes/no***

Meaning	- time after which the RTS line will be zeroed after having sent data to the station. The parameter has meaning only when the control by means of the RTS line is switched on.
Default value	- 10.
Parameter:	
number	- time in milliseconds.

☑ ***Ignore_Source_Address =yes/no***

Meaning	- each packet sent by the station includes the station address. The station address is verified by the controller. In case of incompatibility to the station number it is rejected. Setting yes will cause the controller to stop the sender address verification.
Default value	- no.

EXAMPLE 1

```
[ASMEN]
.....
```

```
MAC=MACMAT,2,COM2
....


[MAC]
Auto_Sync=60
Max_time_difference=10
```

In the example above the station named MAC connected to the COM2 port is defined. The synchronization of computer and station clocks will be performed every 1 minute. If the difference is at least 10 seconds, then the synchronization of clocks occurs.

EXAMPLE 2

```
[ASMEN]
.....
MAC1=MACMAT,1,COM2
MAC2=MACMAT,2,COM2
MAC3=MACMAT,3,COM2
MAC4=MACMAT,4,COM3
MAC5=MACMAT,5,COM3
MAC6=MACMAT,6,COM4
....


[MACMAT]
;Default value for all stations
baud=19200


[MACMAT:3]
;Default values for stations connected to the COM3 port
baud=9600


[MAC6]
Auto_Sync=0
```

In the example above stations with names from MAC1 to MAC6 are defined. The stations MAC1, MAC2 and MAC3 are connected to the COM2 port. The stations MAC4 and MAC5 are connected to the COM3 port. The station MAC6 is connected to the COM4 port. All serial ports except COM3 will work with a speed of 19200 baud. The COM3 port will work with a speed of 9600 baud. The clock of the station MAC6 will not be synchronized.

Defining the Process Variables

<u>Current Measure Data</u>

Variables allowing to access to current measure data have the following form:

B*n*

Where:

n	- is the number of a datum according to the station specification:

B1 - value of a gas flow counter
B2 - Qn
B3 - Qr
..... etc.

The value of Bn variable is a floating-point number.

The variable B0 is not used.

Access to Registered Values

For measures marked according to the specification with numbers 0/1 and from 2 to 8, the access to their values remembered by the station as registered data or twenty-four-hours data (for a flow counter) is possible. The value of a registered variable is the value remembered by the station in the last registration period. An access to older measures is possible by the B type archiving. The registered variables have the following form:

R0 - gas flow counter (in the end of last twenty-four hours)
R2 - Qn
R3 - Qr
. .
R8 - rez2

Access to the list of alarms by using the number of the successive alarm in the list:

Variables allowing to access to the list of alarms have the following form:

A*n.type*

where:

n	- alarm no.
type	- type of information on an alarm according to the table below.

Table 26. Type of Information on an Alarm.

Type Name	Meaning	Type of Received Value
c, code	alarm code (according to the station documentation)	integer number (1 byte)
v, val	increase of counter value in time of alarm duration	floating-point number (4 bytes)
sec0	the second of alarm beginning	integer number (1 byte)
m0, min0	the minute of alarm beginning	integer number (1 byte)
h0, hour0	the hour of alarm beginning	integer number (1 byte)
day0	the day of alarm beginning	integer number (1 byte)
mon0	the month of alarm beginning	integer number (1 byte)
y0,year0	the year of alarm beginning	integer number (2 bytes)
sec1	the second of alarm end	integer number (1 byte)
m1, min1	the minute of alarm end	integer number (1 byte)
h1, hour1	the hour of alarm end	integer number (1 byte)
day1	the day of alarm end	integer number (1 byte)
mon1	the month of alarm end	integer number (1 byte)
y1,year1	the year of end alarm	integer number (2 bytes)

Access to the list of alarms by means of a code of alarms:

Variables allowing to access to the list of alarms by means of an alarm code have the following form:

E*n* or E*n.type*

where:

n	- alarm code according to the documentation;
type	- type of information about the alarm according to the table presented above.

Variable E allows to access to the information about the alarm with a given code. If the list of alarms does not include the code of a required alarm, then a value of 0 (integer type – 1 byte) is returned. If the list contains many alarms with a given code, then the information about the alarm which occurred as latest of all is returned. If the variable type was omitted, then a value of 1 is returned if the alarm with a given code is active, and a value of 0 otherwise. If the variable type is given, then the value from the table presented above is returned.

Access to the List of Alarms as Bit Mask

The variable has the form:

EB*n*

where:

n	- number of byte 0-31.

By means of the EB variable it is possible to read information about active alarms in groups of 8 alarms:

EB0	- alarms with codes 0- 7
EB1	- alarms with codes 8-15
EB2	- alarms with codes 16-23
...	
EB31	- alarms with codes 248-255

The variable value is an integer number of 1 byte length. Individual bytes of a variable value are assigned to adequate alarms. If the bit is set, then the alarm corresponding with it is active. The EB type variable allows to bind the alarms with bit strategy of identifying the alarms of **asix**.

Access to Twenty-Four-Hours Data

The variable has the form:

D*n*

where:

n	- is a number of data in accordance to the station documentation.

The D0 variable has the same meaning as the R0 variable.

Access to Statistical Data

The MACMAT controller enables to access to statistical data referring to the quantity of transmitted data and quantity of errors of the transmission. Variables allowing to access to statistical data have the following form (see: the following table).

Table 27. Variables Allowing to Access to Statistical Data.

Address	Meaning	Type of Received Value
SBS	quantity of sent bytes	Integer number (4 bytes)
SBR	quantity of sent bytes	Integer number (4 bytes)
SFS	quantity of sent frames	Integer number (4 bytes)
SFR	quantity of sent frames	Integer number (4 bytes)
SPE	quantity of errors of parity	Integer number (4 bytes)
SFE	(frame errors) quantity of frame errors	Integer number (4 bytes)
SOE	quantity of errors of overrun	Integer number (4 bytes)
SLE	quantity of line errors (the amount of parity errors, frames, overrun and etc.)	Integer number (4 bytes)
STE	quantity of timeout errors	Integer number (4 bytes)
SPRE	quantity of protocol errors	Integer number (4 bytes)
SCE	quantity of checksum errors	Integer number (4 bytes)
SLGE	quantity of logical errors (lack of data in the controller, incorrect address etc.)	Integer number (4 bytes)
SERR	amount of all errors (SLE, STE, SPRE, SCE and SLGE). The saving of any value into ERR variable is causes zeroing the following variables: SBS, SBR, SFS, SFR, SPE, SFE, SOE, SLE, STE, SPRE, SCE and SLGE.	Integer number (4 bytes)
TSBS	quantity of sent bytes (from the beginning of driver working)	Integer number (4 bytes)
TSBR	quantity of received bytes (from the beginning of driver working)	Integer number (4 bytes)
TSFS	quantity of sent frames (from the beginning of driver working)	Integer number (4 bytes)
TSFR	quantity of received frames (from the beginning of driver working)	Integer number (4 bytes)
TSPE	quantity of errors of the parity (from the beginning of driver working)	Integer number (4 bytes)
TSFE	quantity of the frame errors (from the beginning of driver working)	Integer number (4 bytes)
TSOE	quantity of errors of overrun (from the beginning of driver working)	Integer number (4 bytes)
TSLE	quantity of errors of the line (the amount of errors of parity, frames, overrun and etc.)	Integer number (4 bytes)
TSTE	quantity of timeout errors (from the beginning of driver working)	Integer number (4 bytes)
TSPRE	quantity of protocol errors (from the beginning of driver working)	Integer number (4 bytes)
TSCE	quantity of checksum errors (from the beginning of driver working)	Integer number (4 bytes)
TERR	amount of errors determined with following variables: TSLE, TSTE, TSPRE, TSCE, TSOE	Integer number (4 bytes)

Access to Historical Data (for the Version with Access to Historical Data)

The MACMAT controller enables the archive ASPAD module to access to historical data for variables from B1 to B8 and R0 and from R2 to R8:
- for variables R0 and B1, daily data are read;
- for variables B2 and B3, registered data of flow increment are read.; these data are scaled so that they express a flow per 1 hour; the data are scaled in the basis of registration frequency read from the station;
- for variable B4 to B8 and R2 to R8, adequate registered historical data are read.

Cooperation with AsComm Manager of Connections

In order to use the AsComm manager of connections to establish connections with MACMAT stations you should place the following record in the application INI file:

AsComm = Yes

This record is placed in the [MACMAT] or [MACMAT:n] section, where *n* signifies the number of the serial port declared in the [ASMEN] section.

In case of cooperation with the AsComm module, the port number from the channel declaration is used to create the name, which is used by the driver to exchange data with the AsComm module. The name has a form of *MacMAT-n*, where *n* is the number of the serial port from the channel declaration.

EXAMPLE

```
[ASMEN]
.....
MAC=MACMAT,2,COM2
....
[MACMAT]
.....
AsComm = Yes
....
```

The example above defines driver of the name MacMAT-2 as a client of AsComm. This name is also the name of the section, in which parameters of connections established by the AsComm module such as modem name, telephone number, etc. are placed. The description of parameters, which should be placed in such section, may be found in the manual of the AsComm module. You should notice that the number of the serial port may refer, but it has not, to the physical serial port. This number signifies the real serial port only when in the section (named [MacMAT-n]) for parameterization of connections established by the AsComm module other records defining truly used port (e.g. modem name) are not given.

An example of configuring the AsComm module for dial-up connections is given below:

```
[ASMEN]
.....
MAC=MACMAT,2,COM2
....
[MACMAT]
.....
AsComm = Yes
....

[MACMAT-2]
switched_line = Yes
Modem =Sportster Flash
Interval = 5m
Max_Connection_Time=2m
```

```
Number = 12345678
```

In order to establish connections the modem Sportster Flash will be used. Connections will be established every 5 minutes with the number 12345678. Maximal connection duration time is equal to 2 minutes.

Definition of Alternative Ports

The MACMAT driver allows using alternative serial ports in case of communication problems occurring when using the basic port (that appears in the definition of the logical channel). The parameter declaring the alternative port may be inserted into the [MacMAT] or [MacMAT:n] section and has the following form:

Alt_port = COMm, metod_of_switching_to_alternative_port, metod_of_switching_to_basic_port

The parametr defines the COMm serial port used in case of communication problems in the COMn port (determined in the logical channel definition). The parameter may occur only one time (only one alternative port is permissible). The COMm alternative port is not allowed to be declared as the basic port in the other ASMEN channel definition.

Switching to the alternative port occurs after fulfilling the following condition:

Errors_number[*/time_period*]

When a number of missed reading attempts determined by *Errors_number* occurs in *time_period*, switching to the alternative port follows. The quantity of errors also includes the quantity of the transmission operation repetitions performed by the driver. That means, that if the occurrence of 3 errors is the condition of switching and the number of repetitions is 5, switching may occur during the time of current request performing and this request has a chance to be performed correctly through using the alternative channel. If not, the request will be performed with an error status and the switching to the alternative port will occur during performing next ASMEN requests. The *time_period* parameter may be omitted - in such case, the switching will occur after *Errors_number* appearance. The *time_period* parameter is passed in seconds.

The return to using the basic port occurs after a number of seconds determined by the *metod_of_switching_to_basic_port* parameter since the moment of switching to the alternative channel. It doesn't mean that the modem connection will be realized all the time (if the alternative channel is that connection). This connection will be served like so far, it means it will be disconnected as a result of the AsComm module parameterization or when all ASMEN requests are realized. The parameters of the serial port should be defined in the [MacMAT] section if they are supposed to be other than the ones applied in case of the basic port.

EXAMPLE

An example of the alternative channel parametrization:

```
[ASMEN]
...
MAC1=MACMAT,220,COM1
```

```
...
[MACMAT]
Baud = 9600
;switching to the alternative channel, after 3 following errors have occurred, and return switching after
;2 minutes
Alt_Port = COM2, 3, 120
Alt_Port = COM2, 15/60, 120
; Or switching to the alternative port after 15 errors having occurred during 1 minute. The return switching as above.
```

1.32. MBUS - Driver of M-BUS Protocol

Driver Use

The M-Bus standard was developed as a standard for communication with heat counters and is the most widespread in this branch. This protocol was tested and developed in connection with MULTICAL heat meters of KAMSTRUP A/S.

Driver Configuration

The driver is configured in the line defining the logical channel in the **[ASMEN]** section of the application INI file. The definition of the channel is as follows:

Channel_name=Unidriver, mbus, *Driver_parameters*

Driver_parameters have the following form:

Name=*value*[,*Name*=*value*] ....

or

[*section_name*]

where:

section_name	- section name in the application INI file where all the driver parameters are written (each in a separate line).

EXAMPLE

MBUS1=UniDriver, mbus,[*section_name*]

[*section_name*]
Adres=48
...

☑ ***Address =number***

Meaning	- determines the address of a given M-BUS device. The parameter is an obligatory parameter.
Default value	- lack.
Parameter:	
number	- the parameter is a number from a range 1 to 250.

Alarmn =alarm_numbers

Meaning	- is a set of parameters with names from *Alarm0* to *Alarm7*. Each parameter determines a number of **asix** system alarm, which will be generated by the driver after appearance of an analogous alarm in the M-BUS device. The meanings of alarms generated by the M-BUS device is specified by the manufacturer.
Default value	- lack.

Alarm_Code =alarm_number

Meaning	- the parameter determines the alarm number in the **asix** system which is generated by the driver after a lost of connection with the M-BUS device.
Default value	- lack.

Port = COMn[:baud[:word[:parity[:stop]]

Meaning	- determines a serial port used for the communication and transmission parameters.
Default value	- COM*n*:2400:8:even:1.
Parameter:	
n	- serial port number,
baud	- transmission speed,
word	- word length,
parity	- parity (none, even, odd, mark, space),
stop	- number of stop bits.

The parameter *Port* is obligatory. If transmission parameter was omitted, then the default values are taken. The port no. must be always given.

Refresh_Period =number

Meaning	- determines an interval by means of which the driver reads data from the M-BUS device.
Default value	- 15.
Parameter:	
number	- is passed in seconds.

Refresh_Delay =number

Meaning	- the parameter determines the minimal time between successive data readings from the M-BUS device. Some devices (e.g. MULTICAL) needs a considerable time to prepare data. The parameter determines the time necessary for preparing data by the M-BUS device.
Default value	- 12.
Parameter:	
number	- is passed in seconds.

Double_Read =Yes/No

Meaning	- some devices (e.g. MULTICAL) return the data prepared after a previous reading. If the parameter has a value *Yes,* then driver will do two successive reading in order to obtain the most actual data.

Default value	- Yes.

☑ ***Invalid_Statuses =number,number,...***

Meaning	- the M-BUS device send a status byte with measured data. Each of bits of this byte determines a specified data value. The parameter determines what status bits make the received data invalid.
Default value	- +1,+2,+3,+4,+5.
Parameter:	
number,number,...	- the parameter has a form of set of bit numbers separated with the '+' character. The least significant bit has the number 1. The default value (1,2) means that the data are found incorrect if the device signals the error „application busy" (1), „application error" (2), „power drop" (3), „constant error" (4) and „temporary error" (5). The manufacturer may define additional statuses.

☑ ***Log =file_name***

Meaning	- the parameter value is a name of file where diagnostic information is written. The parameter may be used only for diagnostic purposes.
Default value	- lack.

☑ ***timeout =number***

Meaning	- the parameter determines a maximal waiting time for an answer. The time is expressed in milliseconds.
Default value	- the default value is determined on the ground of transmission parameters according to the M-BUS protocol specification.
Parameter:	
number	- time in miliseconds.

☑ ***Timeout2=number***

Meaning	- the parameter determines the maximal waiting time to receive one character.
Default value	- the default value is determined on the ground of transmission parameters according to the M-BUS protocol specification.
Parameter:	
number	- time in miliseconds.

☑ ***Dump=file_name***

Meaning	- the parameter value is a name of the file where the data will be written. The parameter may be used only for diagnostic purposes.
Default value	- lack.

Defining the Process Variables

The set of variables handled by the driver may be divided into several groups:

- sequential variables, i.e. the variables the address of which is the number of the datum transferred by the M-BUS device;
- variables, the address of which contains the name of measured variable;
- variables, which enable to read the manufacturer's data and other variables.

Sequential Variables

The definition of sequential variables needs the sequence in which the M-BUS device sends measured data to be known.

The address of sequential variables is as follows:

P*n*

where *n* is a variable order number.

Addressing by Means of Variable Name

The variable address is as follows:

Name [.*Un*][.*Tn*][.*Sn*]

where:

Name	- name of a measured variable;
Un	*n* - unit number (if omitted, then it is assumed to be equal to 0). The unit number is applied when the device consists of several units;
Tn	*n* - tariff number (if omitted, then it is assumed to be equal to 0);
Sn	*n* - cell number to store historical data (storage) (if omitted, then it is assumed to be equal to 0).

It is allowed to use the following names (see: the following table).

Table 28. Set of Acceptable Measured Variable Names.

Name	Meaning
ACCESSNUMBER	Next number of data reading
ACTDURATION	Duration time in seconds
AVGDURATION	Duration time in seconds
BAUDRATE	Transmission speed
BUSADD	Device address
CREDIT	Credit
CUSTOMERLOC	Localization of client
DEBIT	Debit
DIGINPUT	Digital input
DIGOUTPUT	Digital output
EIDENT	Extended identification
ELCURRENT	Current in amperes
ENERGY	Energy
FABRNO	Factory number
FLOWTEMP	Temperature
FVERSION	Firmware version
HVERSION	Equipment version
MANUFACTURER	Manufacturer
MASS	Mass
MASSFLOW	Masse flow
MEDIUM	Code of measured medium
MODEL	Model
ONTIME	Time from turning on
OPERTIME	Work time
PARAMSETID	Identification of parameters
POWER	Power
PRESS	Pressure
RESPDELAY	Delay of device response
RETTEMP	Return temperature
SVERSION	Software version
TEMPDIFF	Difference of temperatures
TIMEPOINT	Time stamp
VOLUME	Volume
VOLFLOW	Volume flow
XVOLFLOW	External volume flow
XTEMP	External temperature

Addressing the Manufacturer's Data

Manufacturer's data are the data, that are not described in the protocol definition. To read them, the knowledge of manufacturer's data structure of a given device is necessary.

The address of a manufacturer's datum is as follows:

M*position*.*length*

where:

position	- byte number in the manufacturer's data block, from which a given value begins; the first byte has the number 0;
length	- data length in bytes.

The driver assumes that the manufacturer's data are expressed in the BCD code.

Access to Unit Symbol of Measurement

For sequential variables and variables addressed with a variable name it is allowed to define variables that return the symbol of a measured physical unit (e.g. Wh for energy). To do it, you should add /UNIT to the variable address e.g. ENERGY/UNIT. As a conversion function you should define NOTHING_TEXT. In order to display the unit on technological diagram you can use the object STRING.

Other Data

Data transferred by the M-BUS device may be provided with a header. Variables allowing to access the data in a header:

Table 29. Set of Variables Allowing to Access the Data in a Header.

Address	Meaning	Type
H.IDENT	Identifier of device	DWORD
H.MANUFACTURER	Manufacturer code	
H.VERSION	Version	DWORD
H.MEDIUM	Medium code	BYTE
H.ACCESSNO	Next number of reading	BYTE
H.STATUS	Data status	BYTE

Data Status

Devices operating according to the M-BUS protocol make available the datum of 1 byte length, the individual bits of which determine the device status in the following way (see: ***Table 29***):

Table 30. Data Statuses for M-BUS Devices.

Number	Meaning	Byte Number
1	application engaged	0
2	application error	1
3	voltage drop	2
4	constant error	3
5	temporary error	4
6	error specific for the device	5
7	error specific for the device	6
8	error specific for the device	7

Statuses with the successive number from 1 to 5 cause all the data sent by the device to be cancelled by the driver i.e. they take the status *bad datum*. It doesn't apply to the data contained in the header described in point Other Data. This default operation of the driver may be changed by the parameter *Invalid_Statuses* (see: *Driver Configuration*). The datum containing the device status may be read by the variable H.STATUS described in point Other Data. The fird column of the above table determines the byte number, which corresponds to the specified status, in the variable.

Not all the devices make available the statuses 1 and 2. The meaning of statuses 6,7 and 8 is defined by the manufacturer.

1.33. MEC - Driver of MEC07 and MEC08 Heat Meter Protocol

Driver Use

The driver is designed for data exchange between the **asix** system and MEC07 and MEC08 heat meters manufactured by the Instytut Techniki Cieplnej (Institute of Thermal Technology) in Łódź. Data exchange is executed by means of a serial interface in the RS-232 standard. For switching serial line between counters a multiplexer, controlled by RTS (switching to the first channel) and DTR (switching to the next channel) lines, is used.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel operating according to the MEC protocol is given below:

channel_name=MEC, *port* *[,max_no]*

where:

MEC	- driver name,
port	- name of the serial port connected to the multiplexer of MEC heat meters, e.g. COM1,
max_no	- optional number of multiplexer channels to which the MEC heat meters are connected (by default 5).

EXAMPLE

An exemplary declaration of transmission channel CHANNEL used for communication with MEC heat meters connected to the multiplexer channels numbered from1 to 8. Data exchange is executed through the serial port COM2.

```
CHANNEL = MEC, COM2, 8
```

Types of Process Variables

In the driver one type of process variables is defined:

V	- value of transferred measurement.

All process variables are of FLOAT type and may be only read. A time stamp is given by a heat counter and sent with values of process variables within a common telegram. The time stamp is given with the accuracy of one minute.

The syntax of symbolic address used for variables belonging to the MEC driver channel is as follows:

V<nrFabr>.<nrOdb>.<index>

where:

nrFabr	- factory number of the MEC heat meter;
nrOdb	- number of the receiver from which the measurement is transferred; allowed value is 1 or 2;
index	- variable number in a table of variables transferred from the meter.

The list of indexes and variables corresponding with them are given in tables 1 and 2.

Raw values of variables read from heat counters should be converted following the conversion function NOTHING_FP or FACTOR_FP. The column *Factor* in both tables contains factors **A** of the conversion function FACTOR_FP for measurements with known conversion method of raw value (factors **B** are equal to 0). For the other measurements the calculation method of real value of the measurement should be agreed with the user of heat meters.

Table 31. Variables Transferred from MEC07.

Variable Name	Index	Factor
Factory number	1	1
Number of receiver	2	1
Status	3	1
Time of exceeding Max	4	
Time of exceeding Min	5	
Time of range exceeding	6	
Total time	7	0.1
Time of damage	8	
Total masse	9	10
Number of exceeding	10	
	11	
Number of feed decays	12	
Actual power	13	0.001
Current of break. PDP0	14	
Flux of masse	15	0.01
Ordered power	16	0.001
Delta of energy	17	
Qmax (exceeding)	18	
Qmin (exceeding)	19	
Qsum (total energy)	20	
Duration time of exceeding	21	
Max power of exceeding	22	
Energy of exceeding	23	
Volume of exceeding.	24	
V/Vz of exceeding.	25	
Air temperature (Tpow)	26	0.01
Delta T (Trozn)	27	0.01
Average air temp. (Tsrpow)	28	0.01
Delta T average (Tsrroz)	29	0.01
Temp.of feed. (Tzas)	30	0.01

Table 32. Variables Transferred from MEC08.

Variable Name	Index	Factor
Factory number	1	1
Receiver number	2	1
Status	3	1
Pressure	4	0.001
Time of exceeding.	5	
Time in exceeding min.	6	
Time in feed.	7	
Time of range exceeding	8	
Time of range exceed. of condens.	9	
Total time	10	0.1
Time of damage	11	
Close time	12	
Close time of feed	13	
Enthalpy	14	1
Enthalpy of condens.	15	0.1
Total masse of condens..	16	0.1
Total masse	17	0.1
Number of exceedings	18	
Number of feed decays	19	
Steam power	20	0.01
Current of break. PDP0	21	
Current of break. PDP1	22	
Power of condens.	23	0.01
Ordered power	24	0.01
Energy in exceeding	25	
Energy in exceeding min.	26	
Energy in feed..	27	
Energy of condens.	28	0.1
Total energy	29	0.1
Duration time of exceeding.	30	
Max power of exceeding.	31	
Energy of exceeding.	32	
Flux of condens. masse	33	0.01
Flux of masse	34	0.01
Overheat (delta T)	35	0.1

EXAMPLE

Examples of declarations of variables.

```
A0, time 7502,              V7502.1.00, CHANNEL, 17, 1, DATETIME_MEC
A1, pressure 7502,          V7502.1.04, CHANNEL, 1,   1, FACTOR_FP, 0.001,
0
A2, temp. of feed 7502,     V7502.1.37, CHANNEL, 1,   1, FACTOR_FP, 0.1, 0
A3, total time 8502,        V8502.1.07, CHANNEL, 1,   1, FACTOR_FP, 0.1, 0
A4, factory no 7502,        V7502.1.01, CHANNEL, 1,   1, NOTHING_FP
```

Driver Configuration

Configuring the driver is executed by using the separate section named **[MEC]**. By means of this section it is possible to declare the following positions.

☑ ***LoG_FILE = file_name***

Meaning - allows to define a file to which all diagnostic driver messages and information about contents of telegrams received by the driver are written. If the item does not define the full path, the log file will be created in the

	current directory. The log file should be used only in the stage of the **asix** start-up.
Default value	- log file is not created.
Defining	- manual.

☑ ***LOG_FILE_SIZE = number***

Meaning	- allows to specify the log file size in MB.
Default value	- 1MB.
Defining	- manual.

☑ ***LOG_OF_TELEGRAMS=[YES|NO]***

Meaning	- item allows to write to the log file (declared bye use of the item LOG_FILE) the contents of telegrams received by the driver. Writing the contents of telegrams to the log file should be used only in the stage of the **asix** system start-up.
Default value	- NO.
Defining	- manual.

☑ ***DATA_VALID_TIME = number***

Meaning	- for each MEC counter the time of the last reading is checked. If it exceeds the declared value, then the data from the actually read MEC counter receive an error status.
Default value	- 1min.
Defining	- manual.

☑ ***STROBE_TIME = number***

Meaning	- allows to determine the duration time (in milliseconds) of the RTS and DTR strobe while controlling the switching of multiplexer channels.
Default value	- 60 ms.
Defining	- manual.

☑ ***NUMBER_OF_REPETITIONS = number***

Meaning	- allows to determine the number of repetitions for all channels of multiplexer.
Default value	- 1 repetition.
Defining	- manual.

1.34. MELSECA - Driver of MITSUBISHI MELSEC-A PLC Protocol

Driver Use

The MELSECA driver is used for data exchange between **asix** computers and the A1SJ71C24-R2 communication processor of MITSUBISHI MELSEC-A PLCs. The transmission is executed by means of serial interfaces by using standard serial ports of an **asix** system computer.

The cooperation of **asix** with the controller by using the MELSECA protocol does not require any controller's program adaptation.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel working according to the MELSECA protocol is given below:

logical_name=MELSECA,*type,pc_cpu,port,[baud,character,parity,stop]*

where:

type	- set of realized commands: ACPU or AnCPU;
pc_cpu	- PC CPU number; in case of connection point-point it should be given ff (*self PC CPU number*),
port	- serial port name;
baud	- transmission speed in baud;
character	- number of bits in a transmitted character;
parity	- parity check type (even,odd,none);
stop	- number of stop bits.

Parameters *baud*, *character*, *parity*, *stop* and *buffer* are optional. In case of omitting them the default values are taken as follows:

- transmission speed - 9600 Bd,
- number of bits in a character - 8,
- parity check type - parity check (none),
- number of stop bits - 1.

EXAMPLE

An example of declaration of transmission channel operating according to the MELSECA protocol is given below:

```
CHAN1=MELSECA,AnCPU,ff,COM1,9600,8,even,1
```

The transmission channel with logical name CHAN1 has defined the following parameters:

- MELSECA protocol,
- set of commands AnCPU,
- connection point-point (self PC CPU number),
- port COM1,
- transmission speed of 9600 Bd,
- transmitted character length - 8 bits,
- parity check,
- one stop bit.

Addressing the Process Variables

The syntax of symbolic address which is used for variables belonging to the MELSECA driver channel is as follows:

VARIABLE_TYPE variable_index

where:

VARIABLE_TYPE	- string identifying the variable type in the MELSECA protocol,
variable_index	- index within a given type.

The following symbol of types of process variables are allowed (in columns on the right side the range of variable indexes for the ACPU and AnCPU sets of commands are given).

	ACPU	AnCPU
X	- Input X 0 - 7FF	0 - 7FF
Y	- Output Y 0 - 7FF	0 - 7FF
M	- Internal relay M.0 - 2047	0 - 8191
L	- Latch relay L 0 - 2047	0 - 8191
S	- Step relay S 0 - 2047	0 - 8191
B	- Link relay B 0 - 3FF	0 - 0FFF
F	- Annunciator F 0 - 255	0 - 2047
TS	- Timer (contact) T0 - 255	0 - 2047
TC	- Timer (coil) T 0 - 255	0 - 2047
TN	- Timer (present value) T0 - 255	0 - 2047
CS	- Counter (contact) C 0 - 255	0 - 1023
CC	- Counter (coil) C 0 - 255	0 - 1023
CN	- Counter (present value) C 0 - 255	0 - 1023
MS	- Special relay M9000 - 9255	9000 - 9255
D	- Data register D0 - 1023	0 - 6143
W	- Link register W0 - 3FF	0 - 0FFF
R	- File register R 0 - 8191	0 - 8191
DS	- Special register D9000 - 9255	9000 - 9255

The variable index for types X, Y, B and W is given in hexadecimal form, at the same time the indexes beginning with a letter should be preceded by digit 0, e.g. a correct declaration of input no. E has a form of X0E (declaration XE will be rejected as wrong).

Indexes of the other types are given in decimal format.

EXAMPLE

X0A2	- value of input no. A2
D1010	- value of register D no. 1010

All process variables are treated as 16-bit numbers.

A correct operation of a communication processor A1SJ71C24-R2 requires appropriate setting of switches SW04 - SW12 and MODE (work mode) on the front panel. The MODE switch should be unconditionally set to position 1, because the MELSECA driver is based on the dedicated protocol with number 1. The SW04 switch should be set to ON if the application executes controls (state ON of the switch allows to write data in RUN mode). The SW12 switch should be set to ON (counting and verification of a checksum). The state of switches SW05 - SW11 should be set according to transmission parameters given in the item declaring a MELSECA transmission channel:

(transmission speed, number of characters in a word,number of stop bits, manner of parity check).

The cable connecting a communication processor A1SJ71C24-R2 with an **asix** system computer should be made according to the pattern given for the connection with a device which does not use signals DTR/DTS for transmission check *(see: chapter 4.5 External Wiring of documentation 'Computer Link Module type A1SJ71C24-R2').*

The MELSECA driver is installed as a DLL automatically.

Driver Configuration

Configuration in the **[MELSECA]** section.

☑ ***LOG_FILE = file_name***

Meaning	- allows to define a file where all diagnostic messages of the MELSECA driver and information about the contents of telegrams received and sent by the MELSECA driver will be written.
Default value	- by default, the contents of telegrams are not written to the log file.

☑ ***CHECKSUM=YES|NO***

Meaning	- allows to use a checksum in the protocol.
Default value	- YES.

☑ ***LOG_OF_TELEGRAMS=YES|NO***

Meaning	- allows to write to the log file (declared by using the item LOG_FILE) the contents of telegrams sent and received by the MELSECA driver within the reading/writing process variables.
Default value	- NO.

☑ ***LOG_FILE_SIZE=number***

Meaning	- allows to specify the size of log file in MB.
Default value	- 1 MB.

1.35. MEVAS - Driver of MEVAS Analyzers

Driver Use

The MEVAS driver is used for data exchange between MEVAS emission computers and an **asix** system computer. The communication is executed by means of serial interfaces. The driver realizes the protocol described in *„Bedienungsanleitung Rechnerschnittstelle MEVAS (vorlaufige Version 1.00). Telegrammverkehr uber eine serielle Schnittstelle. Stand: 26.03.1993"*.

Declaration of Transmission Channel

A logical channel is a logical connection of a computer and a MEVAS station. The logical channel is defined by placing an appropriate record in the **[ASMEN]** section.

The full syntax of declaration of transmission channel operating according to the MEVAS protocol is given below:

logical_name=MEVAS,*COMn*, *Mevas_Address*

where:

COMn	- number of the serial port to which the network of MEVAS controllers is connected;
Mevas_Address	- number identifying the MEVAS emission computer; the number is assigned on the stage of the MEVAS emission computer parameterization.

Each defined channel may have its own section, the name of which is the logical name of the channel. A given COMn port may also have its own section named **[MEVAS:n]**. Values defined in such section become the default values for individual stations. The default values for individual serial interfaces are taken from the section named **[MEVAS]**. Transmission parameters by means of a serial interface cannot be placed in sections concerning individual stations, i.e. they may be located only in sections [MEVAS] and [MEVAS:n].

Driver Configuration

The statement of items placed in the application INI file for the MEVAS driver.

☑ ***baud = number***

or

☑ ***bps= number***

Meaning	- determines transmission speed.
Default value	- 9600
Parameter**:**	
number	- number passed in Bd.

☑ ***parity = parity_number***

Meaning	- determines parity.	
Default value	- n	
Parameter:		
number	- allowed values:	
	n	- no parity bit,
	o	- odd parity check,
	e	- even parity check,
	m	- mark,
	s	- space.

☑ ***retries = number***

Meaning	- number of repetitions of missed reading operations from a MEVAS station.
Default value	- 3

☑ ***stop_bits = number***

Meaning	- determines number of stop bits.
Default value	-1
Parameter:	
number	- allowed values are 1 and 2.

☑ ***word = number***

or

☑ ***word_length = number***

Meaning	- word length.
Default value	- 8
Parameter:	
number	- allowed values are from interval 5 to 8.

☑ ***time_out = number***

or

☑ ***timeout = number***

Meaning	- waiting time for DMS285 answer.
Default value	- 10000
Parameter:	
number	- number passed in milliseconds.

☑ ***bad_data_statuses = status1,status2,...,statusN***

Meaning	- determines numbers of data status, for which the data are found invalid. Status 13 (no data) causes that the data is always treated as invalid, independently of parameter value.
Default value	- 0,13
Parameter:	
	- format: status1, status2, ..., statusN or the character - (lack of the *bad data* status, except 13).

☑ ***log = log_file***

Meaning	- parameter determines the name of file to which additional diagnostic information will be written.
Default value	- lack

☑ ***alarm_code = alarm_number***

Meaning	- parameter determines a number of alarm generated by the driver in case of loss and re-establishing a connection with the station. The value of -1 (by default) causes that alarms are not generated. In a situation of connection loss, the following number specifying the cause of connection loss is transmitted together with an alarm code:
	0 – complete lack of any answer from the station;
	1 – timeout;
	2 – line errors (frame, parity, overrun errors);
	3 – checksum errors;
	4 – other errors;
	5 - MEVAS was reset;
	6 - timeout on the MEVAS side;
	7 – checksum error on the MEVAS side.
	This number determines the end status of last attempt to establish the connection.

Default value	- -1

☑ ***simulation = number***

Meaning	- if the parameter value is 1, then the driver works in simulation mode and does not communicate with the station. Values of all variables are random.

Default value	- 0

☑ ***Refresh1,...,Refresh10= number***

Meaning	- parameters determine how often the driver has to transfer to the MEVAS station requests of preparing a new data set for later reading. Each parameter corresponds to a definite variable group. The parameter *Refresh1* refers to all variables. The parameter has a form of two numbers. The first number determines frequency of request sending, the second one determines a shift in time of request sending. For instant, if 60s, 10s is given, then requests of preparing new data may be sent at hours: 12:00:10, 12:01:10, 12:02:10 etc. The parameter determines only the maximal frequency of request sending. If the driver does not receive requests to read new data from the other components of the **asix** system (ASMEN), then requests to prepare new data are not sent to the MEVAS station. If the parameter *Refresh1* and one of parameters *Refresh2-Refresh10* has a non-zero value, then the requests to prepare a data collection, defined by this last parameter, will be sent at moments fulfilling criteria defined by both the parameters simultaneously i.e. with a frequency equal to the minimal value of both the parameters. If the parameter *Refresh1* has value 0,0, then corresponding to them other parameters should have non-zero values. Data read from the MEVAS station – except the D28 data (integrals) – receive a time stamp transferred by the MEVAS station at the moment of receiving a request to prepare a new data collection.
Default value	- Refresh1=60s,10s (other parameters have values 0,00).
Parameter:	
number	- both the numbers have a format *nnn[s|m|g|h]* where *nnn* determines the time and the letter designation determines the base of time (second, minute, hour, hour respectively). If time unit designation is omitted, then the second is assumed.

☑ ***Round_28= yes/no***

Meaning	- if the parameter has a value yes, then the time of the D28 data is rounded up to a full hour. The rounding refers only to data with time in form *hh:59:00*. Other time values are not rounded. To switch off the rounding one should give no as the parameter value.
Default value	- 15

☑ ***Round_28_Time = yes/no***

Meaning	- time in minutes determining the time rounding range of hour integrals. If the integral time is included in the

	range: the nearest full hour +/- parameter value, then the integral time will be rounded to the nearest full hour. By example, if the parameter value is 15 minutes, then times of integrals from the range 9:45:00 to 10:15:00 will be rounded to 10:00:00. The rounding occurs if the value of parameter *Round_28* is yes. If the parameter value is 0, then the rounding does not occur.
Default value	- 15

☑ ***Time_Read = number***

Meaning	- the parameter determines a time interval in seconds, with which the driver updates the station time. The driver cyclically reads the station time with a given interval. The MEVAS station time is used for determining the time, at which requests to prepare new data should be sent to the MEVAS station.
Default value	- 3600.
Parameter:	
number	- time in seconds.

☑ ***Max_D28_Time = number***

Meaning	- determines the way of driver reaction to lack of the D28 data (integrals) during historical data completing. It is the time of data lack, in seconds, after which the driver will assume that historical data do not exist and transfer such information to write in the archive. The parameter is used only in the situation when the MEVAS station reports a lack of any values referring to a definite channel and the value type (Knr/Wsl). The parameter does not concern a situation when only a part of possible 52 historical values is not available. In this last case it is assumed that such situation will not change later.
Default value	- 4200
Parameter:	
number	- time in seconds.

EXAMPLE

Examples of the MEVAS driver configuration.

Example 1:

```
[ASMEN]
.....
MVS_1=MEVAS,COM2,3
....

[MEVAS:2]
baud=19200
```

In the example above the station named MVS_1 connected to the COM2 port was defined. The transmission speed of 19200 bps will be used. The station has an identifier of 3.

Example 2:

```
[ASMEN]
.....
MVS_1=DMS285,COM1,1
MVS_2=DMS285,COM2,2
MVS_3=DMS285,COM3,1
MVS_4=DMS285,COM4,1
MVS_5=DMS285,COM5,4
MVS_6=DMS285,COM6,5
....

[MEVAS]
;Default values for all stations
baud=19200
Invalidity_Status= 1, 6, 14
[MEVAS:3]
;Default values for stations connected to the COM3 port
baud=9600

[MVS_2]
Invalidity_Status= 5

[MVS_3]
Invalidity_Status= 0,13
```

In the example above stations with names form MVS_1 to MVS_6 connected to ports from COM1 to COM6 are defined. All serial ports except COM3 will work with a speed of 19200 baud. The COM3 port will work with a speed of 9600 baud. All stations except MVS_2 and MVS_3 stations will use invalidity statuses 1, 6 and 14. The MVS_2 station uses a value of 5 as an invalidity status. The MVS_3 status does not use any invalidity status – setting parameter „-„ was necessary to change the default values set in the [MEVAS] section.

Time Stamp

Other data than D28 are transferred by driver to the **asix** system together with a time received from the MEVAS station while executing a request from the MEVAS station to prepare new data for reading. The frequency of sending request is specified by parameters *Refresh1*,..., *Refresh10*.

For data D28 (integrals) the MEVAS stations send a set of maximally 52 values. Each of these values is provided with its own time, which usually has a form of hh:59 (for 1-hour cycle of integration). This time is rounded up by driver to a full hour. The rounding may be turned off by means of parameter *Round_28*. Range of rounding defines a parameter *Round_Time_28*.

Although each D28 datum has its own time, for the variables of this type it is also necessary to send to the MEVAS station a request to prepare new data and if the parameter *Refresh1* has a value of 0,0 then the parameter *Refresh5* must have a not zero value in order to read data correctly (historical data too).

Defining the Process Variables

The variable definition is based on the MEVAS protocol description.

List of all types of variables is given in the end of this point.

name [.arg1[.arg2[.arg3]]]

where:

argn	- may be: number, number preceded by text or text.

Square brackets [and] include parts, which may be absent in a variable definition.

The variable name may be a number of the value described in an appropriate query (of D type) of the communication protocol with the MEVAS system. The number may be preceded by a letter D. Variables defining data of other queries of the protocol are the numbers preceded by the query type e.g. X1 to X5 and S1 to S5.

A record <*n..m*> signifies a numerical value from a range n to m. A vertical line „|" signifies that it is allowed to choose one of text on either side of the line.

In the table below all the types of variables are placed. The column *Number of parameter Refresh* specifies, which of parameters from *Refresh2* to *Refresh10* concerns a given variable. The value of variables from X1 to X10 is a time of preparing a specific data group by the MEVAS station as a result of the lately sent request. Writing any value to these variables causes sending to the MEVAS station a request of preparing new data for reading.

If the *Status* column contains Yes, then a given variable is transferred to the **asix** system with the status defined by the parameter *Invalidity_Statuses*. Such variable is accessed by the MEVAS station together with the status defined by the protocol. This status is converted to a numerical value and compared with values defined by the parameter *Invalidity_Statuses*. If the value of converted status is compatible to one of values defined by the parameter *Invalidity_Statuses*, then the data is regarded as invalid. The data with a status of 13 (no data) is always regarded as invalid.

Table 33. Format of Variable Name.

Format of Variable Name	TYPE	Number of Parameter *Refresh*	Status	Record	Example
[D]1.[Knr|ch]<1..48>	WORD	2			D1.Knr1
[D]2.[Knr|ch]<1..48>	FLOAT	2	Yes		D2.Knr1
[D]2.[Knr|ch]<1..48>.F|VAL	FLOAT	2			D2.Knr1.F
[D]2.[Knr|ch]<1..48>.SS|STS|STA	WORD	2			D2.Knr1.SS
[D]3.[Knr|ch]<1..48>	FLOAT	2	Yes		D3.Knr1
[D]3.[Knr|ch]<1..48>.[F|VAL]	FLOAT	2			D3.Knr1.F
[D]3.[Knr|ch]<1..48>.SS|STS|STA	WORD	2			D3.Knr1.SS
[D]4.[Knr|ch]<1..48>	FLOAT	2	Yes		D4.Knr1
[D]4.[Knr|ch]<1..48>.F|VAL	FLOAT	2			D4.Knr1.F
[D]4.[Knr|ch]<1..48>.S|STS|STA	WORD	2			D4.Knr1.S
[D]4.[Knr|ch]<1..48>.int|i	WORD	2			D4.Knr1.int
[D]5.[Knr|ch]<1..48>	FLOAT	2	Yes		D5.Knr1
[D]5.[Knr|ch]<1..48>.F|VAL	FLOAT	2			D5.Knr1.F
[D]5.[Knr|ch]<1..48>.S|STS|STA	WORD	2			D5.Knr1.S
[D]5.[Knr|ch]<1..48>.int|i	WORD	2			D5.Knr1.int
[D]6.[Knr|ch]<1..48>	FLOAT	2	Yes		D6.Knr1
[D]6.[Knr|ch]<1..48>.F|VAL	FLOAT	2			D6.Knr1.F
[D]6.[Knr|ch]<1..48>.S|STS|STA	WORD	2			D6.Knr1.S
[D]6.[Knr|ch]<1..48>.int|i	WORD	2			D6.Knr1.int
[D]7.[Knr|ch]<1..48>.[int|i|val]	WORD	2			D7.Knr1.int
[D]8.[Knr|ch]<1..48>	FLOAT	2	Yes		D8.Knr1
[D]8.[Knr|ch]<1..48>.F|VAL	FLOAT	2			D8.Knr1.F
[D]8.[Knr|ch]<1..48>.S|STS|STA	WORD	2			D8.Knr1.S
[D]8.[Knr|ch]<1..48>.int|i	WORD	2			D8.Knr1.int
[D]9.[Knr|ch]<1..48>	FLOAT	2	Yes		D9.Knr1
[D]9.[Knr|ch]<1..48>.F|VAL	FLOAT	2			D9.Knr1.F
[D]9.[Knr|ch]<1..48>.S|STS|STA	WORD	2			D9.Knr1.S
[D]9.[Knr|ch]<1..48>.int|i	WORD	2			D9.Knr1.int
[D]10.[Knr|ch]<1..48>	FLOAT	2	Yes		D10.Knr1
[D]10.[Knr|ch]<1..48>.F|VAL	FLOAT	2			D10.Knr1.F
[D]10.[Knr|ch]<1..48>.S|STS|STA	WORD	2			D10.Knr1.S
[D]10.[Knr|ch]<1..48>.int|i	WORD	2			D10.Knr1.int
[D]11.[Knr|ch]<1..48>	FLOAT	2	Yes		D11.Knr1
[D]11.[Knr|ch]<1..48>.F|VAL	FLOAT	2			D11.Knr1.F
[D]11.[Knr|ch]<1..48>.S|STS|STA	WORD	2			D11.Knr1.S
[D]11.[Knr|ch]<1..48>.int|i	WORD	2			D11.Knr1.int
[D]12.[Knr|ch]<1..48>.[F|VAL]	FLOAT	2			D12.Knr1.F
[D]12.[Knr|ch]<1..48>.Ag|cnt	WORD	2			D12.Knr1.Ag
[D]13.[Knr|ch]<1..48>.[F|VAL]	FLOAT	2			D13.Knr1.F
[D]13.[Knr|ch]<1..48>.Ag|cnt	WORD	2			D13.Knr1.Ag
[D]14.[Knr|ch]<1..48>.[F|VAL]	FLOAT	2			D14.Knr1.F
[D]14.[Knr|ch]<1..48>.Ag|cnt	WORD	2			D14.Knr1.Ag
[D]15.[Knr|ch]<1..48>.[F|VAL]	FLOAT	2			D15.Knr1.F
[D]15.[Knr|ch]<1..48>.Ag|cnt	WORD	2			D15.Knr1.Ag

Table 34. Format of Variable Name (continuation).

Format of Variable Name	TYPE	Number of Parameter *Refresh*	Status	Record	Example
[D]16.[Knr|ch]<1..48>.[F|VAL]	FLOAT	2			D16.Knr1.F
[D]16.[Knr|ch]<1..48>.Ag|cnt	WORD	2			D16.Knr1.Ag
[D]17.[Knr|ch]<1..48>.[F|VAL]	FLOAT	2			D17.Knr1.F
[D]17.[Knr|ch]<1..48>.Ag|cnt	WORD	2			D17.Knr1.Ag
[D]18.[Knr|ch]<1..48>.[I|VAL]	WORD	6			D18.Knr1.I
[D]19.[Knr|ch]<1..48>.[F|VAL]	FLOAT	6			D19.Knr1.F
[D]20.[Knr|ch]<1..48>.[F|VAL]	FLOAT	6			D20.Knr1.F
[D]21.[Knr|ch]<1..48>	DWORD	2			D21.Knr1
[D]21.[Knr|ch]<1..48>.val|j|jjjjmm|jjjjjjmm	DWORD	2			D21.Knr1.val
[D]21.[Knr|ch]<1..48>.t|ttttmm|tttttmm	DWORD	2			D21.Knr1.t
[D]24.[Bnr|Bl]<1..96>.[Bsl|Typ]<1..3>.KL0|CL0|0	WORD	3			D24.Bnr1.Bsl1.KL0
[D]24.[Bnr|Bl]<1..96>.[Bsl|Typ]<1..3>.KL1|CL1|1	WORD	3			D24.Bnr1.Bsl1.KL1
[D]24.[Bnr|Bl]<1..96>.[Bsl|Typ]<1..3>.KL2|CL2|2	WORD	3			D24.Bnr1.Bsl1.KL2
[D]24.[Bnr|Bl]<1..96>.[Bsl|Typ]<1..3>.KL3|CL3|3	WORD	3			D24.Bnr1.Bsl1.KL3
[D]24.[Bnr|Bl]<1..96>.[Bsl|Typ]<1..3>.KL4|CL4|4	WORD	3			D24.Bnr1.Bsl1.KL4
[D]24.[Bnr|Bl]<1..96>.[Bsl|Typ]<1..3>.KL5|CL5|5	WORD	3			D24.Bnr1.Bsl1.KL5
[D]24.[Bnr|Bl]<1..96>.[Bsl|Typ]<1..3>.KL6|CL6|6	WORD	3			D24.Bnr1.Bsl1.KL6
[D]24.[Bnr|Bl]<1..96>.[Bsl|Typ]<1..3>.KL7|CL7|7	WORD	3			D24.Bnr1.Bsl1.KL7
[D]24.[Bnr|Bl]<1..96>.[Bsl|Typ]<1..3>.KL8|CL8|8	WORD	3			D24.Bnr1.Bsl1.KL8
[D]24.[Bnr|Bl]<1..96>.[Bsl|Typ]<1..3>.KL9|CL9|9	WORD	3			D24.Bnr1.Bsl1.KL9
[D]24.[Bnr|Bl]<1..96>.[Bsl|Typ]<1..3>.KL10|CL10|10	WORD	3			D24.Bnr1.Bsl1.KL10
[D]24.[Bnr|Bl]<1..96>.[Bsl|Typ]<1..3>.KL11|CL11|11	WORD	3			D24.Bnr1.Bsl1.KL11
[D]24.[Bnr|Bl]<1..96>.[Bsl|Typ]<1..3>.KL12|CL12|12	WORD	3			D24.Bnr1.Bsl1.KL12
[D]24.[Bnr|Bl]<1..96>.[Bsl|Typ]<1..3>.KL13|CL13|13	WORD	3			D24.Bnr1.Bsl1.KL13
[D]24.[Bnr|Bl]<1..96>.[Bsl|Typ]<1..3>.KL14|CL14|14	WORD	3			D24.Bnr1.Bsl1.KL14
[D]24.[Bnr|Bl]<1..96>.[Bsl|Typ]<1..3>.KL15|CL15|15	WORD	3			D24.Bnr1.Bsl1.KL15
[D]24.[Bnr|Bl]<1..96>.[Bsl|Typ]<1..3>.KL16|CL16|16	WORD	3			D24.Bnr1.Bsl1.KL16

Table 35. Format of Variable Name (continuation).

Format of Variable Name	TYPE	Number of Parameter *Refresh*	Status	Record	Example
[D]24.[Bnr|Bl]<1..96>.[Bsl|Typ]<1..3>.KL17|CL17|17	WORD	3			D24.Bnr1.Bsl1.KL17
[D]24.[Bnr|Bl]<1..96>.[Bsl|Typ]<1..3>.KL18|CL18|18	WORD	3			D24.Bnr1.Bsl1.KL18
[D]24.[Bnr|Bl]<1..96>.[Bsl|Typ]<1..3>.KL19|CL19|19	WORD	3			D24.Bnr1.Bsl1.KL19
[D]24.[Bnr|Bl]<1..96>.[Bsl|Typ]<1..3>.KL20|CL20|20	WORD	3			D24.Bnr1.Bsl1.KL20
[D]24.[Bnr|Bl]<1..96>.[Bsl|Typ]<1..3>.KL21|CL21|21	WORD	3			D24.Bnr1.Bsl1.KL21
[D]24.[Bnr|Bl]<1..96>.[Bsl|Typ]<1..3>.KL22|CL22|22	WORD	3			D24.Bnr1.Bsl1.KL22
[D]24.[Bnr|Bl]<1..96>.[Bsl|Typ]<1..3>.KL23|CL23|23	WORD	3			D24.Bnr1.Bsl1.KL23
[D]24.[Bnr|Bl]<1..96>.[Bsl|Typ]<1..3>.KL24|CL24|24	WORD	3			D24.Bnr1.Bsl1.KL24
[D]24.[Bnr|Bl]<1..96>.[Bsl|Typ]<1..3>.KL25|CL25|25	WORD	3			D24.Bnr1.Bsl1.KL25
[D]24.[Bnr|Bl]<1..96>.[Bsl|Typ]<1..3>.KL26|CL26|26	WORD	3			D24.Bnr1.Bsl1.KL26
[D]24.[Bnr|Bl]<1..96>.[Bsl|Typ]<1..3>.KL27|CL27|27	WORD	3			D24.Bnr1.Bsl1.KL27
[D]24.[Bnr|Bl]<1..96>.[Bsl|Typ]<1..3>.KL28|CL28|28	WORD	3			D24.Bnr1.Bsl1.KL28
[D]24.[Bnr|Bl]<1..96>.[Bsl|Typ]<1..3>.KL29|CL29|29	WORD	3			D24.Bnr1.Bsl1.KL29
[D]24.[Bnr|Bl]<1..96>.[Bsl|Typ]<1..3>.KL30|CL30|30	WORD	3			D24.Bnr1.Bsl1.KL30
[D]24.[Bnr|Bl]<1..96>.[Bsl|Typ]<1..3>.KL31|CL31|31	WORD	3			D24.Bnr1.Bsl1.KL31
[D]25.[Bnr|Bl]<1..96>.[Bsl|Typ]<1..3>.Kl|cl<1..31>	WORD	3			D25.Bnr1.Bsl1.Kl1
[D]26.[Bnr|Bl]<1..96>	DWORD	3			D26.Bnr1
[D]26.[Bnr|Bl]<1..96>.TOT|TOTAL|h|hhhhmm|hhhhhmm	DWORD	3			D26.Bnr1.TOT
[D]26.[Bnr|Bl]<1..96>.ACT|A|aaaamm|aaaaamm	DWORD	3			D26.Bnr1.ACT
[D]26.[Bnr|Bl]<1..96>.CNT|u|uuu	WORD	3			D26.Bnr1.CNT
[D]27.[Bnr|Bl]<1..96>	DWORD	3			D27.Bnr1
[D]28.[Wnr]<1..52>.[Knr|ch]<1..48>.Wsl<1..3>	FLOAT	5	Yes		D28.Wnr1.Knr1.Wsl1
[D]28.[Wnr]<1..52>.[Knr|ch]<1..48>.Wsl<1..3>.F|VAL	FLOAT	5			D28.Wnr1.Knr1.Wsl1.F

Table 36. Format of Variable Name (continuation).

Format of Variable Name	TYPE	Number of Parameter *Refresh*	Status	Record	Example
[D]28.[Wnr]<1..52>.[Knr|ch]<1..48>.Wsl<1..3>.TIM|TIME	WORD	5			D28.Wnr1.Knr1.Wsl1.TIM
[D]28.[Wnr]<1..52>.[Knr|ch]<1..48>.Wsl<1..3>.SEC	WORD	5			D28.Wnr1.Knr1.Wsl1.SEC
[D]29.<1..16>.day	WORD	4			D29.1.day
[D]29.<1..16>.year	WORD	4			D29.1.year
[D]29.<1..16>.YSEC	DWORD	4			D29.1.YSEC
[D]29.<1..16>.DSEC	DWORD	4			D29.1.DSEC
[D]30	DWORD	9			D30
[D]31.s	WORD	Lack			D31.s
[D]31.p	WORD	Lack			D31.p
[D]32.[Egnr]<1..100>.Lnr|no	WORD	Lack	Yes		D32.Egnr1.Lnr
[D]32.[Egnr]<1..100>.Knr|ch	WORD	10	Yes		D32.Egnr1.Knr
[D]32.[Egnr]<1..100>.Hk	WORD	10	Yes		D32.Egnr1.Hk
[D]32.[Egnr]<1..100>.Gk	WORD	10	Yes		D32.Egnr1.Gk
[D]32.[Egnr]<1..100>.[F|VAL]	FLOAT	10	Yes		D32.Egnr1.F
[D]32.[Egnr]<1..100>.time|t	DWORD	10	Yes		D32.Egnr1.time
[D]33.[Knr|ch]<1..48>.[F|VAL]	FLOAT	2			D33.Knr1.F
[D]34.[Knr|ch]<1..48>.[F|VAL]	FLOAT	2			D34.Knr1.F
[D]35.[Knr|ch]<1..48>	FLOAT	2	Yes		D35.Knr1
[D]35.[Knr|ch]<1..48>.F|VAL	FLOAT	2			D35.Knr1.F
[D]35.[Knr|ch]<1..48>.S|STS|STA	WORD	2			D35.Knr1.S
[D]35.[Knr|ch]<1..48>.int|i	WORD	2			D35.Knr1.int
S1.start	WORD	N/A			S1.start
S1.end	WORD	N/A			S1.end
S2.start	WORD	N/A			S2.start
S2.end	WORD	N/A			S2.end
S3.start	WORD	N/A			S3.start
S3.end	WORD	N/A			S3.end
S4.start	WORD	N/A			S4.start
S4.end	WORD	N/A			S4.end
S5.start	WORD	N/A			S5.start
S5.end	WORD	N/A			S5.end
X1	DWORD	N/A		Yes	X1
X2	DWORD	N/A		Yes	X2
X3	DWORD	N/A		Yes	X3
X4	DWORD	N/A		Yes	X4
X5	DWORD	N/A		Yes	X5
X6	DWORD	N/A		Yes	X6
X7	DWORD	N/A		Yes	X7
X8	DWORD	N/A		Yes	X8
X9	DWORD	N/A		Yes	X9
X10	DWORD	N/A		Yes	X10

Table 37. List of Data Statuses.

Number	Status Character According to Protocol	Description Following the Protocol
0	A	Anlage AUS – Device OFF
1	S	Storung EIN – Failure ON
2	W	Wartung – Maintenance
3	w	Wartung manuell – Manual maintenance
4	T	Test
5	P	Plausibilitat AUS – Plausibility OFF
6	M	Verrechnungsfehler – Allocation error
7	E	Ersatzwertsteuerung EIN – Replacement value control ON
8	s	Ersatzwertsteuerung Storung – Failure of replacement value control
9	u	ungultig Anfahrbetrieb – invalid startup
10	U	ungultig (durch Anlage-AUS oder Klassenblockweschel) – invalid (by device OFF or class block changes)
11	*	nich belegt – not occupied
12	?	unknown status
13	********	No data (only for D28 and D32)

Historical Data

An access to historical data is possible for the D28 type. The data are available from the current day beginning (maximum 52 values). The data for the last hour of the previous day are available only in the period 23:59-00:05 (circa). It means that a pause in communication in this period results an irreparable loss of values for the last hour of a day. The current values of the data (integral) should be obtained by using a D28 variable with the parameter *Wnr1*.

1.36. MODBUS - Driver of MODBUS/RTU Protocol for MASTER Mode

Driver Use

The MODBUS driver is used for data exchange with controllers or devices operating according to the MODBUS protocol. The transmission is executed by means of serial interfaces with the use of standard serial ports of an **asix** system computer.

The cooperation of **asix** with the controller by using the MODBUS protocol does not require any controller's program adaptation for data exchange.

(While implementing the MODBUS protocol the RTU mode, which enables a larger capacity of the interface, was used).

he driver has the following data types implemented:

HR	(holding registers),
IR	(input registers),
CS	(coil status),
IS	(input status).

and the following functions of the MODBUS protocol:

Read Coil Status	(function 01),
Read Input Status	(function 02),
Read Holding Registers	(function 03),
Read Input Registers	(function 04),
Force Single Coil	(function 05),
Preset Single Register	(function 06),
Force Multiple Coils	(function 15),
Preset Multiple Registers	(function 16).

Declaration of Transmission Channel

The full syntax of declaration of transmission channel working according to the MODBUS protocol is givn below:

logical_name=MODBUS,*id,port,[baud,character,parity,stop, max_i/o,max_register]*

where:

id	- device identifier (slave id);
port	- serial port name (max number of handled ports: 32);
baud	- transmission speed in baud; max transmission speed is equal to 115 kBd;
character	- number of bits in a transmitted character;
parity	- parity check type (even,odd,none);
stop	- number of stop bits;
max_i/o	- maximal number of inputs/outputs, the value of which may be transferred by devices within one cycle (max 127*16 i/o states);
max_register	- maximal number of registers, the state of which may be transferred by the device within one cycle (max 127 registers).

Parameters *baud*, *character*, *parity*, *stop*, *max_i/o*, *max_register* and *buffer* are optional. In case of omitting them the following default values are taken:

- transmission speed - 9600 Bd,
- number of bits in a character - 8,
- type of parity check - parity check,
- number of stop bits - 1,
- default number of inputs/outputs - 16,
- default number of registers - 4.

EXAMPLE

An example of declaration of transmission channel working according to the MODBUS protocol is givn below:

CHAN1=MODBUS,2,COM1,9600,8,even,1,40,16

The transmission channel with the logical name CHAN1 has the following parameters defined:

- MODBUS protocol using a serial interface;
- device identifier (slave id) 2;
- port COM1;
- transmission speed of 9600 Bd;
- transmitted character length - 8 bits;
- parity check;
- one stop bit.

Addressing the Process Variables

The syntax of symbolic address used for variables belonging to the MODBUS driver channel is as follows:

VARIABLE_TYPE variable_index

where:

variable_type	- string identifying the variable type in the MODBUS protocol;

variable_index	- variable index within a given type.

The following symbols of types of process variables are allowable:

CS	- Coil Status	(0X reference);
IS	- Input Status	(1X reference);
HR	- Holding Register	(4X reference);
IR	- Input Register	(3X reference);
HRL	- 2 successive Holding Registers treated as a double word in INTEL format;	
HRF	- 2 successive Holding Registers treated as a floating-point number in INTEL format;	
HRLM	- 2 successive Holding Registers treated as a double word in MOTOROLA format;	
HRFM	- 2 successive Holding Registers treated as a floating-point number in MOTOROLA format;	
IRL	- 2 successive Input Registers treated as a double word in INTEL format;	
IRF	- 2 successive Input Registers treated as a floating-point number in INTEL format;	
IRLM	- 2 successive Input Registers treated as a double word in MOTOROLA format;	
IRFM	- 2 successive Input Registers treated as a floating-point number in MOTOROLA format.	

EXAMPLES

CS22	- Coil 22
IS197	- Input 197
HR118	- Holding Register 118
IR25	- Input Register 25

The MODBUS driver is loaded as a DLL automatically.

Considering the appearance of controllers using 32-bit registers, the MODBUS driver was extended with the support of 32-bit registers HR and IR:

HR32L	- 32-bit register HR of DWORD type (requires a calculation function based on DWORD, e.g. NOTHING_DW);
HR32F	- 32-bit register HR of FLOAT type (requires a calculation function based on FLOAT, e.g. NOTHING_FP);
IR32L	- 32-bit register of DWORD type (requires a calculation function based on DWORD, e.g. NOTHING_DW);
IR32F	- 32-bit register of FLOAT type (requires a calculation function based on FLOAT, e.g. NOTHING_FP).

EXAMPLE

```
X1, register HR no 10 as DWORD, HR32L10,  CHAN32, 1, 1, NOTHING_DW
X2, register IR no 20 as FLOAT, IR32F20,  CHAN32, 1, 1, NOTHING_FP
```

Driver Configuration

The MODBUS protocol driver may be configured by means of the **[MODBUS]** section placed in the application INI file. Individual parameters are transferred in separate items of the section. Each item has the following syntax:

item_name=[number [,number]] [YES|NO]

LOG_FILE=file_name

Meaning	- the item allows to define a file where all diagnostic messages of MODBUS driver and information about the contents of telegrams received and sent by the MODBUS driver will be written. If the item does not define the full path, then the log file will be created in the current directory. The log file should be used only while the **asix** start-up.
Default value	- by default, the log file is not created.

LOG_OF_TELEGRAMS=YES|NO

Meaning	- the item allows to write to the log file (declared by using the item LOG_FILE) the contents of telegrams sent and received by the MODBUS driver within the reading/writing process variables. Writing the contents of telegrams to the log file should be used only while the **asix** start-up.
Default value	- by default, the contents of telegrams are not written to the log file.

NUMBER_OF_REPETITIONS=number

Meaning	the item allows to define maximal number of trials to do the command in case of transmission errors.
Default value	- by default, max. 3 repetitions are executed.
Parameter:	
number	- number of repetitions.

NO_ASCOMM=yes/no

Meaning	- determines use of AsComm module to set connection for the MODBUS driver.
Default value	- no.

RECV_TIMEOUT=slave_no,time

Meaning	- the item allows to specify a maximal waiting time for arriving the first character of an answer from a specified remote device. After passage of this time it is assumed that the device under consideration does not work correctly and the transmission session ends with an error.
Default value	- by default, it is assumed that the maximal waiting time for the first character of an answear is equal to 1000 milliseconds.
Parameter:	
slave_no	number of slave placed in the declaration of the transmission channel using the MODBUS protocol;
time	- ber from a range of 100 - 65000 milliseconds.

EXAMPLE

```
[MODBUS]
```

```
RECV_TIMEOUT=2,400
```

☑ ***CHAR_TIMEOUT=slave_no,time***

Meaning	- the item allows to specify a maximal waiting time for arriving the successive character of the answer from a specified remote device. After passage of this time it is assumed that the device under consideration does not work correctly and the transmission session ends with an error.
Default value	- by default, it is assumed that the maximal waiting time for the successive character of the answer is equal to 100 milliseconds.
Parameter:	
slave_no	- slave no. placed in the declaration of transmission channel using the MODBUS protocol;
time	- number from a range of 10 - 2000 milliseconds.

EXAMPLE

```
[MODBUS]
CHAR_TIMEOUT=2,400
```

☑ ***BLOCK_READ=NO/YES[,slave1, slave2, ... slaven]***

Meaning	- the item enables to set an operation mode, in which values of registers and coils are read individually (the function of block data reading is not used). It is valid for ALL the variables supported by the driver.
Default value	- by default, the mode of block data reading is used.
Parameters:	
NO,slave1, slave2, ... slaven	- numbers of slaves the variable values will be read from one by one (without block read);
YES,slave1, slave2, ... slaven	- numbers of slaves the variable values will be read from by means of block read.

☑ ***TRANSMISSION_DELAY=number***

Meaning	- the item allows to declare a time interval between the end of receiving the answer and sending the successive query to the remote device.
Default value	- by default, the item assumes a value of transmission time of 3,5 characters.
Parameter:	
number	- time; the maximal value is equal to 55 ms.

Connection by Means of Modem

EXAMPLE

The MODBUS protocol may also exchange data by means of a modem connection.

The MODBUS driver channel is a client of server AsComm named MODBUS:n, where *n* is a number of the serial port taken from the ASMEN channel definition e.g.

if *channel_name*=modbus,4,com3,...

then the client name is MODBUS:3.

The record given below must be placed in the [MODBUS:n] section for the AsComm program to establish a connection on dial-up links by means of the AsComm program.

Switched_line = Yes

If the modem is connected to an other port than COMn, then you should give the number of this port by means of the parameter *Port* or specify the modem name by means of the parameter *Modem*. You should also give a telephone number and define other parameters required. If MODBUS driver has to communicate with many controllers by means of the same modem, then one should define suitable number of channels assuming the parameter *port* as a virtual transmission channel and place suitable number of sections in the initialization file, by specifying in them an appropriate telephone number.

EXAMPLE

An example of initialization file content.

```
[ASMEN]
....
Chan1 = MODBUS,1,COM11,9600,8,none,1,16,16
Chan2 = MODBUS,1,COM12,9600,8,none,1,16,16

[MODBUS:11]
Switched_line = Yes
Modem = US Robotics
Number = 11111111

[MODBUS:12]
Switched_line = Yes
Modem = US Robotics
Number = 22222222
```

In the example above Chan1 will communicate with a controller placed under the telephone number 11111111, and the Chan2 with a controller placed under the telephone number 22222222. The US Robotics modem will be used. The *Modem* parameter may be replaced by the parameter *Port*, which specifies the number of the serial port to which the modem is connected.

You should notice the given above description of using the MODBUS driver on switched links does not include the modem parameterization. The modem configuration depends on types of used modems.

1.37. MODBUS_TCPIP - Driver of MODBUS_TCP/IP Protocol for OPEN MODBUS/TCP Mode

Driver Use

The driver MODBUS_TCPIP is designed for data exchange between the **asix** system and other computers/devices with use of the MODBUS protocol realized via an Ethernet network with the TCP/IP protocol. The default operating mode of the MODBUS_TCPIP driver is the Open Modbus/TCP mode, developed on the base of specification titled: *"OPEN MODBUS/TCP Specification" Release 1.0, issued 29.03.1999 by the firm Schneider Electric*. The driver allows simultaneous operation in both SLAVE and MASTER modes.

SLAVE Mode

The SLAVE mode consists in executing the read/write commands for the ASMEN data sent from other computers/devices, which are defined as MASTER in the MODBUS network. It is allowed to connect many computers operating as MASTER in the network.

The ASMEN variables are accessed in the SLAVE mode as MODBUS variables from one of the types specified below:

- CS (coil status),
- HR (holding registers),
- IR (input registers).

Declarations of assignment (mapping) of the ASMEN variables to the MODBUS variables are placed by the application designer in text files, which are read by the driver during starting the **asix** system (see: an example on the end of the chapter).

To read the ASMEN variables the following functions of the MODBUS protocol are implemented:

• Read Coil Status	(function 01),
• Read Holding Registers	(function 03),
• Read Input Registers	(function 04).

To write the ASMEN variables the following functions of the MODBUS protocol are used:

• Preset Single Coil	(function 05),
• Preset Single Register	(function 06),
• Preset Multiple Coils	(function 15),
• Preset Multiple Registers	(function 16).

Declaration of Transmission Channel

The declaration of transmission channel for the SLAVE mode is as follows:

channel_name=MODBUS_TCPIP, SLAVE, port [, number]

where:

MODBUS_TCPIP	- driver name;
SLAVE	- operation in the SLAVE mode;
port	- port no. where MASTER computers, which want to connect to the **asix** system, will be listened;
number	- number assigned to an **asix** system computer in the MODBUS network (by default, 1).

NOTE *It is allowed to declare only one transmission channel executing the SLAVE mode.*

SLAVE Driver Configuration

The driver configuration in the **asix** system is performed by means of the separate section named **[MODBUS_TCPIP_SLAVE]** in the application INI file. By using this section it is possible to declare:

- list of clients accepted by the driver;
- mapping the MODBUS variables to the ASMEN variables for individual clients;
- transfer method of the status for individual clients;
- method of refreshing the variables.

☑ ***PROTOCOL_TYPE = <IP_address>, OVATION***

Meaning	- the support of stations operating according to the specification MODBUS RTU requires to use an item.
Parameter:	
IP_address	- address of the station operating as the master.

☑ ***CLIENT_IP_ADDRESS=client_IP_address [,READ_ONLY]***

Meaning	- to the **asix** system only the clients which have an access permission may connect. The clients are declared by CLIENT_IP_ADDRESS items placed in the MODBUS_TCP_SLAVE section.
Default value	- by default, each client may read and write **asix** system variables. It is possible to limit clients access only for reading the variables by using the READ_ONLY option. In this situation an attempt to write is acknowledged by sending a response telegram of the *exception* type with a code 1 which signifies an illegal function.
Parameter:	
client_IP_address	- IP address of the client accepted by the driver;
READ_ONLY	- option allowing the client to read data only.

Number of client declarations not limited.

EXAMPLE

A declaration of the client with IP address 10.10.10.84 with unrestricted access to the **asix** system:

CLIENT_IP_ADDRESS=10.10.10.84

☑ ***CONNECTION_TIMEOUT=client_IP_address,number***

Meaning - the maximal time, which may elapse between successive queries from client (connection timeout), is declared for each connected client. After exceeding the connection timeout, the connection with the client is broken. In case of clients, for which the timeout declaration of is not defined, a default value of 5 minutes is assumed.

Parameter:

client_IP_address	- IP address of a client,
number	- timeout in minutes

EXAMPLE

The declaration of 10-minute connection timeout for the client with the address IP 10.10.10.84:

CONNECTION_TIMEOUT=10.10.10.84, 10

Transferring Status of Variables in Separate Registers and Coils

☑ ***REGISTER_STATUS=client_IP_address***

☑ ***COIL_STATUS=client_IP_address***

Meaning - the MODBUS protocol does not use a status with reference to transferred values of registers and coils. For this reason the status may be joined to the standard MODBUS transfer only in an artificial way. The driver under consideration may transfer the **asix** variable status by placing it in the next element following one which contains the variable value. For registers it is the next register, for coils – the next coil. In case of registers the space for status is sufficient (16 bits), for coils the status must be limited to two states (0 - good , 1 - bad).

By example, if the variable value is transferred as the register 10, then the status is transferred as the register 11. If the coil state is transferred in the bit no. 5, then the coil status is transferred in the bit no. 6.

The applied method allows to send the variable value and its status in the same telegram which ensures the data compactness.

The mode of status transferring is declared individually for each client, separately for registers and for coils. Declarations enable:

- transferring status of registers,

- transferring status of coils.

Default value - by default, the status is not transferred either for registers or for coils.

Parameter:

REGISTER_STATUS	- transferring status of register,
COIL_STATUS	- transferring status of coils,
client_IP_address	- client IP address.

Transferring Status of Variables Without Using Separate Registers or Coils

☑ ***REGISTER_STATUS_ERROR=client_IP_address,number***

Meaning - in case of work without transferring statuses in separate registers or coils, a problem of transferring the information that the variable value is invalid (e.g. errors of communication with data source) arises. The convention is assumed, according to which it is possible to declare the value transferred in case of incorrect status for registers (by default, 0xffff).

Parameter:

client_IP_address	- client IP address;
number	- 16-bit value (HEX), transferred in case of an incorrect variable status.

EXAMPLE

The declaration of transferring the number 0x8000 to the client 10.10.10.84 in case of an error signalized in the status of the variable mapped on MODBUS registers:

```
REGISTER_STATUS_ERROR=10.10.10.84, 0x8000
```

NOTE *In case of coils, because of a binary character of a variable, there is no possibility to transfer the information about the status in such way that it might differ from the correct variable value. Considering it you should also be assumed as a rule that in the work mode without status transferring the registers should be used.*

Declaration of Mapping MODBUS Variables to the ASMEN Variables

Because the MODBUS protocol does not use any names of variables but numbers of registers and coils, it is necessary to prepare configuration files for mapping MODBUS variables to ASMEN variables.

The declaration of mapping a MODBUS variable to an ASMEN variable is as follows:

asix_name, MODBUS_address [, credibility_time]

where:

asix_name	- process variable name of the **asix** system; it must have its equivalent amongst names of process variables in ASMEN files;

MODBUS_address	- type and index of a MODBUS variable, by means of which the process variable value is accessed; depending on the MODBUS variable type, its value is transferred as one bit, one register (a variable of SHORT or USHORT type) or two successive registers (a variable of type FLOAT, LONG or ULONG);
validity_time	- time interval (in seconds), in which the variable value is assumed to be correct. This time is defined as a difference between the time stamp of a query from client and time stamp of an actually accessed variable value. By default, it is equal to 5.

An accepted format of the MODBUS address is as follows:

HR<reg_no> or IR<reg_no> or CS<coil_no>

where:

reg_no	- number of register HR or IR;
coil_no	- number of coil.

The number of MODBUS registers used for transferring the variable value is closely related to the ASMEN variable type:

- for WORD type variables it is one register or one coil;
- for INT16 type variables it is one register;
- for DWORD, LONG or FLOAT type variables it is two registers;

or in case of status transfer:

- for WORD type variables it is two registers or two coils;
- for INT16 type variables it is two registers;
- for DWORD, LONG or FLOAT type variables it is two registers.

The method of register content interpretation for FLOAT, LONG and ULONG type numbers of MODBUS:

First register																Second register															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Mantissa – less significant bits																Z	Exponent														

Figure 1. Format FLOAT (IEEE 745).

First register																Second register															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Z	More significant bits															Less significant bits															

Figure 2. Format LONG.

First register																Second register															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
More significant bits																Less significant bits															

Figure 3. Format ULONG.

EXAMPLE

In ASMEN the variables X_WORD, X_INT, X_DWORD, X_LONG, X_FLOAT are defined as follows:

```
X_WORD,     ED120.2, CHAN1, 1, 1, NOTHING
X_INT,      ED130.2, CHAN1, 1, 1, NOTHING_INT
X_DWORD,    EL140.2, CHAN1, 1, 1, NOTHING_DW
```

X_LONG,	EL150.2, CHAN1, 1, 1, NOTHING_LONG
X_FLOAT,	EG160.2, CHAN1, 1, 1, NOTHING_FP

Mapping the values of these variables to a continuous area of registers beginning from the register HR1 (without status transferring) is as follows:

X_WORD,	HR1
X_INT,	HR2
X_DWORD,	HR3
X_LONG,	HR5
X_FLOAT,	HR7

Mapping the values of these variables to a continuous area of registers beginning from the register HR1 (with status transferring) is as follows:

X_WORD,	HR1
X_INT,	HR3
X_DWORD,	HR5
X_LONG,	HR8
X_FLOAT,	HR11

Declarations of mappings of MODBUS variables to ASMEN variables are placed in text files. Localization of files with d mapping declarations is specified by means of the item:

☑ ***MAP_FILE=client_IP_address,file_name***

Meaning - localization of files with declarations of mappings.

Parameter:

client_IP_address	- client IP address;
file name	- name of a file containing declaration of mappings.

The purpose of connection of a file with a client IP address is to enable individual clients an independent way of mapping.

The number of items with declarations of mapping files is unlimited.

Refreshing Variables Accessed by the Driver

The driver may use one of two strategies of handling the variables exported from the **asix** system.

The strategy I (default) assumes that all the variables, the names of which were declared in the file with mapping declarations, are periodically read by a separate driver thread from the ASMEN cache and placed in a common accessible driver buffer. Threads supporting connection with clients build response telegrams on the basis of this buffer content, without necessity to access to the ASMEN API.

The period of reading from the ASMEN cache to shared driver buffer is configured by means of the item:

☑ ***DATA_BUFFER_UPDATE_PERIOD=number***

Meaning	- enables declaring the period of reading from the ASMEN cache to shared driver buffer.
Default value	- by default the period of reading is equal to 1 second.
Parameter:	
number	- period of reading from the ASMEN cache in seconds.

The strategy II (option) assumes that each thread takes data from the ASMEN cache individually in response to the queries sent by the client. As a result the thread reads from the ASMEN cache only these data, which are required by the client at that moment.

The strategy II should be also activated when **asix** uses the driver only to import data from an other system. In such mode periodical refreshing of contents of driver buffers has no sense.

The strategy II is activated by means of the item:

☑ ***READ_ON_REQUEST = YES***

Values of variables, which are in the ASMEN cache in the moment of executing the read command, may be out-of-date because in the period preceding the client query the refreshing of variables under consideration might be inactive (the **asix** system was just run or none of ASMEN clients was interested in refreshing these variables). Therefore the thread preparing a transfer for the client checks the time stamp actually accessed variables values. If the time stamp value is not contained in the range of *credibility_time* specified in the variable mapping declaration, then the thread reads the variable value directly from the driver and just then returns the answer to the client.

By default, the driver realizes the strategy I with a frequency of updating the content of common accessible buffer equal to 1 second.

☑ ***LOG_FILE=file_name***

Meaning	- the item allows to define a file where all diagnostic messages of the driver in the SLAVE mode and the information about contents of telegrams received/sent by the driver in this mode are written. If the item does not define the full path, then the log file is created in the current directory. The log file should be used only while the **asix** start-up.
Default value	- by default, the log file is not created.
file name	- log file name.

☑ ***LOG_OF_TELEGRAMS =YES|NO***

Meaning	- the item allows to write to the log file the contents of telegrams sent between the driver working in the SLAVE mode and network clients. Writing the contents of telegrams to the log file should be used only while the **asix** start-up.
Default value	- by default, the driver does not write the contents of telegrams to the log file.

☑ ***LOG_FILE_SIZE=number***

Meaning	- the item allows to specify the log file size.
Default value	- by default, the item assumes that the log file has a size of 1 MB.
Parameter:	
number	- log file size in MB.

MASTER Mode

The MASTER mode is the MASTER function implementation in the MODBUS network protocol (in RTU mode) based on ETHERNET with the TCP/IP protocol.

The MASTER mode has implemented the following data types:

CS	- Coil Statuses;
IS	- Input Statuses;
HR	- Holding Registers;
IR	- Input Registers;
HRL	- 2 successive Holding Registers treated as a double word in INTEL format;
HRF	- 2 successive Holding Registers treated as a floating-point number in INTEL format;
HRLM	- 2 successive Holding Registers treated as a double word in MOTOROLA format;
HRFM	- 2 successive Holding Registers treated as a floating-point number in MOTOROLA format;
IRL	- 2 successive Input Registers treated as a double word in INTEL format;
IRF	- 2 successive Input Registers treated as a floating-point number in INTEL format;
IRLM	- 2 successive Input Registers treated as a double word in MOTOROLA format;
IRFM	- 2 successive Input Registers treated as floating-point number in MOTOROLA format.

EXAMPLES

CS22	- Coil 22
IS197	- Input 197
HR118	- Holding Register 118
IR25	- Input Register 25

and the following functions of the MODBUS protocol:

Read Coil Statuses	(function 01),
Read Input Statuses	(function 02),
Read Holding Registers	(function 03),
Read Input Registers	(function 04),
Preset Single Coil	(function 05),
Preset Single Register	(function 06),
Preset Multiple Registers	(function 16 limited to writing a pair of registers).

The MODBUS driver is loaded as a DLL automatically.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel for the MODBUS_TCPIP driver in the MASTER mode is as follows:

channel_name=MODBUS_TCPIP, MASTER, port,IP_address [, number [, binary [, register]]]

where:

MODBUS_TCPIP	- driver name;
MASTER	- MASTER mode;
port	- number of the port by means of which the connection with a slave type device numbered *number* will be executed;
IP_address	- IP address of the device;
number	- number of the slave type device supported in this channel (by default, 1);
binary	- maximal number of binary signals in one query (by default, 32*8);
register	- maximal number of registers in one query (by default, 127).

For each slave type device supported in the MASTER mode a separate declaration of transmission channel is required.

MASTER Mode Configuration

The configuring the MASTER mode is executed by means of the separate section named **[MODBUS_TCPIP_MASTER]**. By means of this section it is possible to declare:

- time to establish a connection on the startup stage,
- timeout of waiting for an answer from the slave type device,
- log file and its size,
- log of telegrams.

☑ ***PROTOCOL_TYPE = <IP_address>, OVATION***

Meaning - the support of stations working according to the specification MODBUS RTU requires to use the item.

Parameter:

IP_address - address of the station being a slave.

Declaration of Startup Time

Establishing connections with slaves is executed on the startup stage of the driver. The default time of startup duration is equal to 3 seconds. It may be modified by means of the item:

☑ ***STARTUP_TIME = number***

Meaning	- startup time of the driver, during which connections are established with slaves.
Default value	- 3 s.
Defining	- manual.

☑ ***RECV_TIMEOUT=client_IP_address, number***

Meaning - for each slave type device a maximal time, which may elapse between sending a query and receiving an answer (so called receiving timeout) is determined. After exceeding the timeout the connection is broken (and re-established). The timeout value is determined individually for each slave type device.

Default value	- in case of slave type devices, for which any timeout declaration is not defined, it is assumed 5 seconds by default.
Defining	- manual.
Parameter:	
client_IP_address	- IP address of a slave type device,
number	- timeout value expressed in seconds.

☑ ***LOG_FILE=file_name***

Meaning	- the item allows to define a file to which all diagnostic messages of the driver generated in the MASTER mode and the information about the contents of telegrams sent/received in this mode by the driver are written. If the item does not define the full path, then the log file is created in the current directory. The log file should be used only while the **asix** start-up.
Default value	- by default, the log file is not created.
Defining	- manual.
Parameter:	
file_name	- log file name.

☑ ***LOG_OF_TELEGRAMS =YES|NO***

Meaning	- the item allows to write to the log file (declared by means of the item LOG_FILE) the contents of telegrams sent between the driver operating in the MASTER mode and slave type devices. Writing the contents of telegrams should be used only while the **asix** start-up.
Default value	- by default, the driver does not write the contents of telegrams to the log file.
Defining	- manual.

☑ ***LOG_FILE_SIZE=number***

Meaning	- the item allows to specify a log file size.
Default value	- by default, the log file size is equal to 1 MB.
Parameter:	
number	- log file size in MB.

1.38. MODBUSSLV - Driver of MODBUS/RTU Protocol for SLAVE Mode

Driver Use

The MODBUSSLV driver is used to access the process variables values of the **asix** system to other systems by means of a serial interface and the MODBUS protocol operating in RTU mode. In such connection the **asix** system operates as a subordinate device (SLAVE) of the MODBUS protocol.

The MODBUSSLV protocol has the following types of variables implemented:

HR	(holding registers),
IR	(input registers),
CS	(coil status).

and the following functions of the MODBUS protocol:

Read Coil Status	(function 01),
Read Holding Registers	(function 03),
Read Input Registers	(function 04),
Force Single Coil	(function 05),
Preset Single Register	(function 06),
Force Multiple Coils	(function 15),
Preset Multiple Registers	(function 16).

The function *Preset Multiple Registers* is limited to write the value of one process variable of the **asix** system.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel operating according to the MODBUSSLV protocol is given below:

logical_channel_name=MODBUSSLV, *number*, *port*, *baud*, *bits*, *parity*, *stop*

where:

MODBUSSLV	- driver name;
number	- number of a remote device of the MODBUS network assigned to **asix**;
port	- name of the serial port;

baud	- transmission speed, max 115 kBd;
bits	- number of bits in a character;
parity	- parity check type;
stop	- number of stop bits.

EXAMPLE

The declaration of the logical channel named CHAN1, which works according to the MODBUSSLV protocol and with parameters as below:

- number - 4
- port - COM1
- transmission speed - 9600 Bd
- number of bits in a character - 8
- parity check type - parity check
- number of stop bits - 1

is as follows:

```
CHAN1 = MODBUSSLV, 1, COM1, 9600, 8, even, 1
```

The MODBUSSLV driver is loaded as a DLL automatically.

Addressing the Process Variables

The syntax of symbolic address which is used for variables belonging to the MODBUSSLV driver channel is as follows:

name, address [,scale] [,WITHOUT_STATUS |WITH_STATUS]

where:

name	- name of the **asix** process variable; it must have its equivalent among process variables declared in ASMEN files;
address	- type and number of the register by means of which the value of process variable is accessed; depending on the process variable type its value is transferred by means of one register (WORD or INT16 type variable) or two successive registers (FLOAT or DWORD type variable); there is a possibility to convert FLOAT type variables to a INT16 type variables;
scale	- *scale* determines a sort of operation ('-' division) and exponent of power of 10 when scalling values (value scalling is currently available for all variable types); it is used in case of converting the value of FLOAT type variables to a INT16 type number with simultaneous scaling (multiplication or division by power of 10), for example;
WITHOUT_STATUS	- variable value is transferred without status;
WITH_STATUS	- variable value is transferred with the status - the status occupies the next register after the register assigned to the variable.

EXAMPLE

A value of the variable X1 is transferred by means of the register HR1. The variable is of FLOAT type and its value is converted to INT16. Before the conversion the variable value is multiplied by 100. Independently of the value of the item WITH_STATUS the status (occupying the RH2 register) is transferred after the variable value.

X1, HR1, 2, WITH_STATUS

The value of the variable X2 is transferred by the HR2 register. Depending on the variable type it occupies only the HR2 register (WORD or INT16 type variable) or HR2 and HR3 registers (FLOAT and DWORD type variable). The variable value is transferred without the status independently of the value of the item WITH_STATUS.

X2, HR2, WITHOUT_STATUS

The value of the X3 variable is transferred by means of the HR3 register (if the variable is of WORD or INT16 type) or HR3 and HR4 registers (if the variable is of FLOAT or DWORD type). If the value of the item WITH_STATUS is YES, then the variable X3 status is transferred in the register HR4 (WORD or INT16 type variable) or HR5 (FLOAT or DWORD type variable).

X3, HR3

Driver Configuration

The MODBUSSLV protocol driver is configured by means of the section **[MODBUSSLV]** placed in the application INI file. Individual parameters are passed in separate items of the section. Each item has the following syntax:

item_name=[number [,number]] [YES|NO]

☑ ***DATA_FILE = file_name***

Meaning	- the item allows to declare a file name in which the declarations of mapping the process variables of **asix** to the registers HR and IR of MODBUS. The section may include many DATA FILE items.
Default value	- lack of default file.

Declaration of Transferring Statuses of Variables

☑ ***WITH_STATUS = YES | NO***

Meaning	- the item allows to define globally the way of transferring the status of **asix** system process variables. The value of the WITH_STATUS item equal to YES signifies that status is sent with the value of the **asix** system process variable. The status is transferred in a register following the last register assigned to the process variable.
Default value	- by default, the value of the item WITH_STATUS is equal to NO.

If the variable is of DWORD type and if the following declaration is given:

```
X1, HR3
```

then the variable status is transferred in the register HR5 (the X1 variable value is transferred in HR3 and HR4 registers).

If the variable X2 is of WORD type and if the following declaration is given:

```
X2, HR3
```

then the variable status is transferred in the register HR4 (the X2 variable value is transferred in the register HR3).

The WITH_STATUS item value equal to NO signifies that the process variable status is not transferred. It is possible to force the global settings by giving the component WITH_STATUS or WITHOUT_STATUS in the declaration of the process variable mapping. The declaration WITH_STATUS signifies that, irrespective of the WITH_STATUS value, the register following registers with the value of the considered variable contains the variable status. The declaration WITHOUT_STATUS signifies that, irrespective of the WITH_STATUS value, the variable status is not transferred.

Casting Variables of FLOAT Type to INT16

☑ ***CAST_TO_INTEGER = YES | NO***

Meaning	- the item equal to YES allows converting the values of FLOAT type to INT16 type. Before converting it is possible to scale the FLOAT value by multiplication by a power of 10. The power value is given optionally in the declaration of the ASMEN variable (component *scale*) mapping.
Default value	- by default, the CAST_TO_INTEGER item value is equal to NO.

The declaration:

```
X1, HR1, -2
```

signifies that the value of X1 process variable will be divided by 100, and then it will be converted to a INT16 type number accessed in the register HR1.

The declaration:

```
X2, HR2, 3
```

signifies that the value of the X2 process variable will be multiplied by 1000, and then it will be converted to a INT16 type number accessed in the HR2 register.

Frequency of Refreshing Contents of Registers

☑ ***DATA_UPDATE_PERIOD = number***

Meaning	- the values of registers are currently updated by reading data from ASMEN. The item determines a refreshing cycle (in seconds).
Default value	- by default, the refreshing cycle is equal to 1 second.

Time of Answer Preparation

☑ ***REPLY_TIMEOUT = number***

Meaning	- time of elaborating an answer (in milliseconds) is controlled by the value of the item REPLY_TIMEOUT. If in the time period passed by the REPLY_TIMEOUT item the driver is not able to prepare data required by the MASTER, then it does not send any reply.
Default value	- by default, the answer preparation time is set to 200 milliseconds.

Maximal Number of Registers Given in a Query

☑ ***NUMBER_OF_REGISTERS_PER_TELEGRAM = number***

Meaning	- the item determines a maximal number of registers that may be asked by MASTER in one query. If the number of registers specified in this item is too large, then the driver sends an answer of the EXCEPTION type with the code 4 (SLAVE DEVICE FAILURE).
Default value	- by default, it is allowed to send queries including up to 128 registers.

☑ ***LOG_FILE=file_name***

Meaning	- the item allows to define a file to which all diagnostic messages of the MODBUSSLV driver are written. If the item does not define the full path, then the log file is created in the current directory. The log file should be used only while the **asix** start-up.
Default value	- by default, the log file is not created.

1.39. MPI - Driver of MPI Protocol for SIMATIC S7 PLCs

Driver Use

The MPI driver is used for data exchange with SIMATIC S7 PLCs by means of the MPI interface. The transmission is executed by serial interfaces in the V24 (RS232C) standard with use of standard serial ports of an **asix** system computer.

Operation of the **asix** system with the SIMATIC S7 PLCs by using the MPI interface does not require any controller's program adaptation in order to perform data exchange.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel operating according to the MPI protocol has the following form:

logical_name=MPI,*port[,PCaddress, MPIaddress,baud,character,parity,stop]*

where:

MPI	- name of the MPI interface driver of SIEMATIC S7 PLCs;
port	- name of the serial port;
PCaddress	- PC address;
MPIaddress	- controller address on the MPI bus;
baud	- transmission speed in bauds;
character	- number of bits in a transmitted character;
parity	- parity check type,
stop	- number of stop bits.

The parameters *PCaddress*, *MPIaddress*, *baud*, *character*, *parity*, *stop* are optional parameters. In case of omitting them the default values are taken:

- transmission speed - 19200 Bd,
- number of bits in a character - 8,
- type of parity check - odd parity check,
- number of stop bits - 1,
- address of S7 controller - 2,
- address of PC - 0.

EXAMPLE

Exemplary items declaring the transmission channel operating according to the MPI protocol are given below:

```
CHAN2=MPI,COM1,0,2,19200,8,odd,1
```

or

```
CHAN2=MPI,COM1
```

The transmission channel named CHAN2 has the following parameters defined:

- port COM1,
- length of transmitted character - 8 bits,
- odd parity check,
- one stop bit.

Addressing the Process Variables

The syntax of symbolic address which is used for variables belonging to the MPI driver channel is as follows:

VARIABLE_TYPE variable_index

where:

VARIABLE_TYPE	- string identifying the variable type in the MPI protocol;
variable_index	- variable index within a given type.

The following symbols of process variable types are allowable (the range of variable indexes is specific for different types of controllers):

EA	- output bytes,
EAW	- output words,
EAD	- output double words,
EE	- input bytes,
EEW	- input words ,
EDI	- 16-byte words in INTEL convention,
EDD	- input double words,
EM	- bytes of flags,
EMW	- words of flags,
EMD	- double words of flags,
EZ	- counter word,
ET	- timer word,
ED	- word in a data block,
EL	- double word in a data block,
ER	- floating-point number in a data block.

In case of data in a data block, after having given the type (EL or ED), you should give the data block number ended with a dot and then the word number.

EXAMPLES

EMW15	- word of flags 15
EE0	- input word 0
EAW8	- output word 8
ED5.3	- word DW3 in data block DB5

The MPI driver is loaded as a DLL automatically.

Driver Configuration

It is possible to set the following item in the **[MPI]** section:

☑ ***DWORD_AS_WORDS=yes/no***

Meaning	- when the value yes is declared, transferring 32-byte data from DB is realized as a transfer of two 16-byte words; when the value no is declared, transferring 32-byte data from DB is realized as a transfer of one double word.
Default value	- no.

1.40. MPS - Driver of MPS Protocol for Power Network Parameter Meters

Driver Use

The MPS driver is used for communication with MPS power network parameter meters made by OBR Metrologii Elektrycznej in Zielona Góra, by means of a serial interface.

Definition of Transmission Channel

The full syntax of declaration of transmission channel operating according to the MPS protocol has the following form:

logical_name=MPS,*controller address,COMn*

where:

COMn	- is the number of the serial port to which the MPS controllers network is connected.

EXAMPLE

An example of the channel definition. In this case, the channel is a logical name of the MPS controller.

```
[ASMEN]
...
MPS1=MPS,1,COM2
...
```

Names of MPS Controller Variables

A variable may be defined by means of the name *FCnn* where *nn* is a number of a variable in the controller according to the description of the controller manufacturer. It is allowed also to use symbolic names. Letter case does not matter.

Table 38. Symbolic Names of MPS Controller Variables.

FC1 - U1	FC30 - Z1
FC2 - U2	FC31 - Z2
FC3 - U3	FC32 - Z3
FC4 - I1	FC33 - R
FC5 - I2	FC34 - R1
FC6 - I3	FC35 - R2

FC7 - P	FC36 - R3
FC8 - Q	FC37 - FI
FC9 - S	FC38 - FI1
FC10 - f	FC39 - FI2
FC11 - cos	FC40 - FI3
FC12 - P15	FC41 - P1s
FC13 - P1	FC42 - P2s
FC14 - P2	FC43 - P3s
FC15 - P3	FC44 - U1m
FC16 - Q1	FC45 - U2m
FC17 - Q2	FC46 - U3m
FC18 - Q3	FC47 - I1m
FC19 - S1	FC48 - I2m
FC20 - S2	FC49 - I3m
FC21 - S3	FC50 - KsU1
FC22 - cos1	FC51 - KsU2
FC23 - cos2	FC52 - KsU3
FC24 - cos3	FC53 - KsI1
FC25 - sin	FC54 - KsI2
FC26 - sin1	FC55 - KsI3
FC27 - sin2	FC56 - Hour
FC28 - sin3	FC57 - Min
FC29 - Z	FC58 - Sec

Definition of Nominal Values

In order to improve the reading of measure values you should define nominal value of voltage, of current and of power (UL, IL, P, Q, S). These values may be given for each station. To do this you should place them in the initialization file in a section named as the logical channel name.

EXAMPLE

```
[ASMEN]
...
MPS1=MPS,1,COM2
MPS2=MPS,2,COM2
...
[MPS1]
UL = 660
IL = 100
...
[MPS2]
UL = 380
IL = 100
...
```

Driver Configuration

Nominal values may be defined in the section named MPS too. The values specified in this section are used as default ones for all controllers i.e. if the section of a given controller does not contain the nominal value, then it is taken from the MPS section.

If all controllers have the same nominal values, then the **[MPS]** section in the application INI file will be enough.

Diagnostic option:

☑ ***Sleep=YES/NO***

Meaning	- yes causes a change of the manner of short time period timing.
Default value	- default value is equal to no.

1.41. MSP1X - Driver of Protocol for MSP-1x ELMONTEX PLCs

Driver Use

The MSP1X driver is used for data exchange between MSP1X PLCs made by ELMONTEX and an **asix** system computer. The communication is executed in the RS485 standard according to the protocol developed for MSP1X PLCs by ELMONTEX (no official specification).

Declaration of Transmission Channel

The full syntax of declaration of transmission channel which operates according to the MSP1X protocol is given below:

channel_logical_name=MSP1X, *group_no, device_no, port [, baud]*

where:

MSP1X	- driver name;
group_no	- number of the group to which the controller belongs;
device_no	- number of a controller within the group;
port	- port name: COM1, COM2 etc.;
baud	- transmission channel (by default, 9600).

Other parameters are default:
- 8 bits in a character,
- without parity check (NONE),
- 1 stop bit.

EXAMPLE

The declaration of the logical channel named CHAN1, which works according the MSP1X protocol and exchanges data with the controller with the number 1, within the group with the number 5, by means of the COM2 port, with default transmission parameters:

```
CHAN1 = MSP1X, 5, 1, COM2
```

The MSP1X driver is loaded as a DLL automatically.

Addressing the Process Variables

The syntax of symbolic address which is used for variables belonging to the MSP1X driver channel is as follows:

<type><index>

where:

type	- variable type,
index	- index within the type.

Symbols of variable types (the raw variable value type is given in parentheses):

AI	- Analog Input	(WORD),
AO	- Analog Output	(WORD),
BI	- Binary Input	(WORD),
BO	- Binary Output	(WORD),
PV	- Preset Value	(WORD),
PD	- Delta Preset Value	(WORD),
IS	- Binary Input Status	(WORD),
OS	- Binary Output Status	(WORD).

EXAMPLE

Examples of declarations of process variables:

```
X4,  analog input nr 1,     AI1,   CHAN1,   1,  1, NOTHING
X5,  binary output nr 15,  BO15,   CHAN1,   1,  1, NOTHING
```

Driver Configuration

The MSP1X protocol driver may be configured by use of the section **[MSP1X]** placed in the application INI file. Individual parameters are transferred in separate items of the section. Each item has the following syntax:

item_name=*[number [,number]] [YES] [NO]*

☑ ***REINITIALIZATION= [YES/NO]***

Meaning	- allows to re-initiate a serial port before each communication session with the controller.
Default value	- NO.
Defining	*- manual.*

☑ ***WRITE_DELAY= number***

Meaning	- declares a time interval (in milliseconds) between the data writing to the controller and the next session of data exchange with the controller.
Default value	- 1200.
Defining	*- manual.*

☑ ***HEADER_DELAY= number***

Meaning	- declares a time interval (in milliseconds) between the characters of a command sent to the controller from the **asix** system.

Default value	- 50.
Defining	*- manual.*

☑ ***DTR_DELAY= number***

Meaning	- declares a time interval (in milliseconds) between sending a command to the controller and setting DTR signaling readiness for data receiving by the **asix** system.
Default value	- 2.
Defining	- manual.

☑ ***UPDATE= number***

Meaning	- declares a time interval (in milliseconds) between the next data reading from the controller to internal driver buffers.
Default value	- 5.
Defining	- manual.

1.42. MUPASZ - Driver of MUPASZ Device Protocol

Driver Use

The MUPASZ driver is used for data exchange between MUPASZ or MUPASZ2000 devices and an **asix** system computer. The communication is performed with use of serial interfaces in the RS232C or RS485 standard.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel which operates according to the MUPASZ protocol is given below:

logical_name=MUPASZ, *number*, *port*, *AlSygOf*, *AlWyłOf*, *AlBlokOf*

where:

MUPASZ	- driver name;
number	- number assigned to the remote device;
port	- name of the serial port;
AlSygOf	- number added to the event number with ***signalization*** executing mode in order to build a unique number of the alarm transferred to the **asix** system;
AlWyłOf	- number added to the event number with ***shutdown*** executing mode in order to build a unique number of the alarm transferred to the **asix** system;
AlBlokOf	- number added to the event number with ***shutdown with lock*** executing mode in order to create a unique number of the alarm transferred to the **asix** system.

EXAMPLE

The declaration of the logical channel named CHAN1, which works according to the MUPASZ protocol and has parameters as below:

- number of remote device - 4
- port - COM1
- number added to the number of event ***signalization*** - 100
- number added to the number of event ***shutdown*** - 200
- number added to the number of event ***shutdown with lock*** - 300

is as follows:

CHAN1=MUPASZ, 4, COM1, 100, 200, 300

The MUPASZ driver is loaded as a DLL automatically.

Addressing the Process Variables

The syntax of symbolic address which is used for variables belonging to the MUPASZ driver channel is as follows:

<variable_type><channel>.<index>

where:

variable_name	- process variable type,
index	- index of a process variable within the type.

Types of process variable:

P	- measurement values (FLOAT),
B	- state of locks arriving with measures (WORD),
L	- values of counters (WORD),
F	- status arriving with measurements and events (WORD).

EXAMPLE

Example of declarations of variables:

```
X1,  current Io,                   P1,   CHAN1,  1,  1,  NOTHING_FP
X2,  I1 accum.,                    P31,  CHAN1,  1,  1,  NOTHING_FP
X3,  state of blockade no. 1,      B1,   CHAN1,  1,  1,  NOTHING
X4,  counter of switch openings,   L1,   CHAN1,  1,  1,  NOTHING
X5,  counter of motor startups,    L25,  CHAN1,  1,  1,  NOTHING
X6,  switch state,                 F1,   CHAN1,  1,  1,  NOTHING
```

Generating Alarms

Numbers of events generated by a remote device have the same range of variation. In order to be able to uniquely determine which device the event comes from, the MUPASZ driver adds to the event number the number specified in the channel declaration as AlSygOf (for signalization), AlWyłOf (for shutdowns) or AlBlokOf (for shutdowns with lock). In such way this number is transferred to the **asix** system as an alarm number.

For some alarms the MUPASZ driver may transfer values coming with events (activation time or current). These values may be read by giving a formatting string (%3.0f) in the definition of an alarm message.

In order to transfer alarms the MUPASZ driver uses the function *AsixAddAlarmGlobalMili( )* by default. The item GLOBAL_ALARMS allows to change default settings and to transfer alarms by means of the function *AsixAddAlarmMili()*.

Driver Configuration

The MUPASZ protocol driver may be configured by use of the **[MUPASZ]** section placed in the application INI file. Individual parameters are transferred in separate items of the section. Each item has the following syntax:

item_name=[number [,number]] [YES|NO]

☑ ***LOG_FILE=file_name***

Meaning	- the item allows to define a file to which all diagnostic messages of the MUPASZ driver and information about contents of telegrams received and sent by the MUPASZ driver will be written. If the item does not define the full path, then the log file is created in the current directory. The log file should be used only while the **asix** start-up.
Default value	- by default, the log file is not created.
Defining	*- manual.*

☑ ***TRANSMISSION_DELAY=number***

Meaning	- the item allows to specify the time period (as a multiple of 10 msec) between successive operations on the MUPASZ bus.
Default value	- by default, the item assumes a value of 1 (10 msec).
Defining	*- manual.*

☑ ***NUMBER_OF_REPETITIONS=number***

Meaning	- the item allows to specify a number of repetitions in case of a transmission error.
Default value	- by default, the item assume a value of 0 (no repetitions).
Defining	*- manual.*

☑ ***DATA_UPDATE=number***

Meaning	- the item allows to define the time period (in seconds), after which the driver should update values of process variables stored in internal driver buffers.
Default value	- by default, the item assumes a value of 1.
Defining	*- manual.*

☑ ***TIME_UPDATE=number***

Meaning	- the item allows to define a time period (in seconds), after which an actual time should be sent to remote devices.
Default value	- by default, the item assumes a value of 60.
Defining	*- manual.*

☑ ***GLOBAL_ALARMS=YES|NO***

Meaning	- the item controls the way of transferring alarms read from remote devices to the **asix** alarm system.
Default value	- by default, the alarms are transferred to the alarm system as global alarms (transferred by means of the function *AsixAddAlarmGlobalMili()*). Setting the GLOBAL_ALARMS item value causes that alarms are transferred to the alarm system by means of the function *AsixAddAlarmMili()*.

Defining - manual.

1.43. MUZ - Driver of Protocol for MUZ Devices

Driver Use

The MUZ driver is used for data exchange between Microprocessor Protecting Devices (MUZ) of the MUZ-RO type made by JMTronik Warsaw and an **asix** system computer.

In the current driver version logical channels are connected with serial ports that service MUZ-RO devices, and not with each of MUZ-RO devices separately.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel which operates according to the MUZ protocol is given below:

channel_name=MUZ, *port*

where:

channel_name	- name of the logical channel;
MUZ	- driver name;
port	- address of the serial port (HEX), which the connection with MUZ-RO devices will be realized by.

The separate logical channel declaration is demanded for each serial port used by the MUZ driver.

Declaration of Device

Declaration of all MUZ-RO devices serviced by the MUZ driver is realized by means of the MUZ position set in the separate [MUZ] section. Each device should have the separate MUZ position.

The MUZ position includes the parameters used for logical channel declaration in the previous driver version. The position syntax is given below:

MUZ=*channel_name,id,type,alarmTxtOff,alarmValOff,var_status,var_control*

where:

MUZ	- position name;
channel_name	- name of the logical channel declared in the [ASMEN] section, wchich the given MUZ-RO device will be serviced by;
id	- number of MUZ-RO in the network;
type	- identifier of the MUZ type (for MUZ-RO it is 7);
alarmTxtOff	- number added to the number of a text alarm transferred from MUZ-RO;
alarmValOff	- number added to the alarm number with the value transferred from MUZ-RO;
var_status	- variable that shows the status of transmission with MUZ-RO;
var_control	- variable that controls the realization of transmission with MUZ-RO.

Syntax of Symbolic Variable Address

The previous symbolic address of the variable has been extended by the prefix containing the MUZ-RO device number. The present address is as follows:

Address.IdMUZ

where:

Address	- the previous address of the symbolic variable;
IdMUZ	- number of the MUZ-RO device, which the variable with the *Address* address is received from.

Symbolic Addresses and Kinds of Variables Used in Data Exchange with MUZ Devices

In the operation of data exchange with MUZ devices, the variables are divided into groups differing with data kind and addressing method. To each group a different symbolic address is assigned. The variables are divided into the following groups:

- values of analog measures;
- binary inputs handled together with measurements (state of single bits is transferred);
- binary inputs handled together with measurements (state of 16 successive SPi variables is transferred);
- rated values and settings of protections;
- binary inputs related to settings of protections;
- variable storing states of events with value;
- variable storing states of text events;
- control variables.

The way of configuring the individual groups of variables is described below.

Table 39. Values of Analog Measures.

Symb. Addr.	Variable in MUZRO	Type of Conversion	Allowed Operation
P1	Current I0	Byte3->Float	Reading
P2	Current I1	Byte3->Float	Reading
P3	Current I2	Byte3->Float	Reading
P4	Current I3	Byte3->Float	Reading
P5	Voltage U0	Byte3->Float	Reading
P6	Voltage U1	Byte3->Float	Reading
P7	Voltage U2	Byte3->Float	Reading
P8	Voltage U3	Byte3->Float	Reading
P9	cos(f1)	Byte4->Float	Reading
P10	cos(f2)	Byte4->Float	Reading
P11	cos(f3)	Byte4->Float	Reading
P12	active power	Byte3->Float	Reading
P13	reactive power	Byte3->Float	Reading
P14	active energy	BCD6->Float	Reading
P15	reactive energy	BCD6->Float	Reading

Table 40. Binary Inputs Handled Together with Measurements (State of Single Bits is Transferred)

8	Variable in MUZRO	Type of Conversion	Allowed Operation
SP1	switch status, closed	Bit (0/1)->Word	Reading
SP2	switch status, closed	Bit (0/1)->Word	Reading
SP3	freely used by client	Bit (0/1)->Word	Reading
SP4	freely used by client	Bit (0/1)->Word	Reading
SP5	local opening of switch	Bit (0/1)->Word	Reading
SP6	freely used by client	Bit (0/1)->Word	Reading
SP7	freely used by client	Bit (0/1)->Word	Reading
SP8	local closing of switch	Bit (0/1)->Word	Reading
SP9	signal for technological protection	Bit (0/1)->Word	Reading
SP10	signal for technological protection 10	Bit (0/1)->Word	Reading
SP11	signal for technological protection 11	Bit (0/1)->Word	Reading
SP12	signal for technological protection 12	Bit (0/1)->Word	Reading
SP13	signal for technological protection 13	Bit (0/1)->Word	Reading
SP14	signal for technological protection 14	Bit (0/1)->Word	Reading
SP15	signal for technological protection 15	Bit (0/1)->Word	Reading
SP16	signal for technological protection 16	Bit (0/1)->Word	Reading
SP17	signal for technological protection 17	Bit (0/1)->Word	Reading
SP18	lock of function of undervoltage protection	Bit (0/1)->Word	Reading
SP19	not used – 0	Bit (0/1)->Word	Reading
SP20	not used – 0	Bit (0/1)->Word	Reading
SP21	not used – 0	Bit (0/1)->Word	Reading
SP22	not used – 0	Bit (0/1)->Word	Reading
SP23	1 – MUZ signals operation of one of protections	Bit (0/1)->Word	Reading
SP24	not used – 0	Bit (0/1)->Word	Reading

Table 41. Binary Inputs Handled Together with Measurements (State of 16 Successive SPi Variables is Transferred)

Symb. Addr.	Variable in MUZRO	Type of Conversion	Allowed Operation
SPW	actual status of variables SP1 (Bit 0) – SP16	WORD ->WORD	Reading
SPW	actual status of variables SP17 (Bit0) – SP24	WORD->WORD	Reading
	(Bits B8 – B15 are filled with zeroes)		

Table 42. Specific Variable in MUZRO.

Symb. Addr.	Variable in MUZRO	Type of Conversion	Allowed Operation
RP1	Number of not read events (operation of protections)		Reading
RP2	event code (1-22) if RP1 ! = 0		Reading

Table 43. Rated Values and Settings of Protections.

Symb. Addr.	Variable in MUZRO	Type of Conversion	Allowed Operation
Z1	Voltage Uzn	Byte3<->Float	Read/Write
Z2	Current Izn	Byte3<->Float	Read/Write
Z3	Utilization factor of network Iw	Byte3<->Float	Read/Write
Z4	Coefficient kodc – current cutter	Byte3<->Float	Read/Write
Z5	Coefficient ki>> - short-circuit protection	Byte3<->Float	Read/Write
Z6	Coefficient ti>> - short-circuit protection	Byte3<->Float	Read/Write
Z7	Coefficient ki> - independent overload protection	Byte3<->Float	Read/Write
Z8	Coefficient ti> - independent overload protection	Byte3<->Float	Read/Write
Z9	Coefficient t1.2 - dependent overload protection	Byte3<->Float	Read/Write
Z10	Coefficient t1.5 - dependent overload protection	Byte3<->Float	Read/Write
Z11	Coefficient t2.0 - dependent overload protection	Byte3<->Float	Read/Write
Z12	Coefficient t3.0 - dependent overload protection	Byte3<->Float	Read/Write
Z13	Coefficient t6.0 - dependent overload protection	Byte3<->Float	Read/Write
Z14	Coefficient ta - dependent overload protection	Byte3<->Float	Read/Write
Z15	Coefficient Ii0> - ground-fault protection	Byte3<->Float	Read/Write
Z16	Coefficient ti0> - ground-fault protection	Byte3<->Float	Read/Write
Z17	Coefficient ku< - undervoltage protection	Byte3<->Float	Read/Write
Z18	Coefficient tu< - undervoltage protection	Byte3<->Float	Read/Write
Z19	Coefficient tt9 - technological protection 9	Byte3<->Float	Read/Write
Z20	Coefficient tt10 - technological protection 10	Byte3<->Float	Read/Write
Z21	Coefficient tt11 - technological protection 11	Byte3<->Float	Read/Write
Z22	Coefficient tt12 - technological protection 12	Byte3<->Float	Read/Write
Z23	Coefficient tt13 - technological protection 13	Byte3<->Float	Read/Write
Z24	Coefficient tt14 - technological protection 14	Byte3<->Float	Read/Write
Z25	Coefficient tt15 - technological protection 15	Byte3<->Float	Read/Write
Z26	Coefficient tt16 - technological protection 16	Byte3<->Float	Read/Write
Z27	Coefficient tt17 - technological protection 17	Byte3<->Float	Read/Write
Z28	Coefficient tiE - impulse time on output E	Byte3<->Float	Read/Write
Z29	Coefficient tiF - impulse time on output F	Byte3<->Float	Read/Write

Table 44. Binary Inputs Related to Settings of Protections.

Symb. Addr.	Variable in MUZRO		Type of Conversion	Allowed Operation
SZ1	on/off	- current cutter	Bit (0/1)<->Word	Read/Write
SZ2	on/off	- short-circuit protection	Bit (0/1)<->Word	Read/Write
SZ3	op/sign.	- overload protection,	Bit (0/1)<->Word	Read/Write
SZ4	on/off	- overload protection,	Bit (0/1)<->Word	Read/Write
SZ5	op/sign.	- overload protection, dependent	Bit (0/1)<->Word	Read/Write
SZ6	on/off	- overload protection, dependent	Bit (0/1)<->Word	Read/Write
SZ7	op/sign.	- ground-fault protection	Bit (0/1)<->Word	Read/Write
SZ8	on/off	- ground-fault protection	Bit (0/1)<->Word	Read/Write
SZ9	op/sign.	- undervoltage protection	Bit (0/1)<->Word	Read/Write
SZ10	on/off	- undervoltage protection	Bit (0/1)<->Word	Read/Write
SZ11	op/sign.	- technological protection 9	Bit (0/1)<->Word	Read/Write
SZ12	dep./indep.	- technological protection 9	Bit (0/1)<->Word	Read/Write
SZ13	op/sign.	- technological protection 10	Bit (0/1)<->Word	Read/Write
SZ14	dep./indep.	- technological protection 10	Bit (0/1)<->Word	Read/Write
SZ15	op/sign.	- technological protection 11	Bit (0/1)<->Word	Read/Write
SZ16	dep./indep.	- technological protection 11	Bit (0/1)<->Word	Read/Write
SZ17	op/sign.	- technological protection 12	Bit (0/1)<->Word	Read/Write
SZ18	dep./indep.	- technological protection 12	Bit (0/1)<->Word	Read/Write
SZ19	op/sign.	- technological protection 13	Bit (0/1)<->Word	Read/Write
SZ20	dep./indep.	- technological protection 13	Bit (0/1)<->Word	Read/Write
SZ21	op/sign.	- technological protection 14	Bit (0/1)<->Word	Read/Write
SZ22	dep./indep.	- technological protection 14	Bit (0/1)<->Word	Read/Write
SZ23	op/sign.	- technological protection 15	Bit (0/1)<->Word	Read/Write
SZ24	dep./indep.	- technological protection 15	Bit (0/1)<->Word	Read/Write
SZ25	op/sign.	- technological protection 16	Bit (0/1)<->Word	Read/Write
SZ26	dep./indep.	- technological protection 16	Bit (0/1)<->Word	Read/Write
SZ27	op/sign.	- technological protection 17	Bit (0/1)<->Word	Read/Write
SZ28	dep./indep.	- technological protection 17	Bit (0/1)<->Word	Read/Write

Table 45. Variables Storing States of Events with Value.

Symb. Addr.	Variable in MUZRO	Type of Conversion	Allowed Operation
ZW8	Current cutter – phase 0	Bit (0/1)->Word	Read/Write
ZW9	Current cutter – phase 3	Bit (0/1)->Word	Read/Write
ZW10	Current cutter – phase 2	Bit (0/1)->Word	Read/Write
ZW11	Current cutter – phases 2 and 3	Bit (0/1)->Word	Read/Write
ZW12	Current cutter – phase 1	Bit (0/1)->Word	Read/Write
ZW13	Current cutter – phases 1 and 3	Bit (0/1)->Word	Read/Write
ZW14	Current cutter – phases 1 and 2	Bit (0/1)->Word	Read/Write
ZW15	Current cutter – phases 1, 2 and 3	Bit (0/1)->Word	Read/Write
ZW16	Short-circuit protection – phase 0	Bit (0/1)->Word	Read/Write
ZW17	Short-circuit protection – phase 3	Bit (0/1)->Word	Read/Write
ZW18	Short-circuit protection – phase 2	Bit (0/1)->Word	Read/Write
ZW19	Short-circuit protection – phases 2 and 3	Bit (0/1)->Word	Read/Write
ZW20	Short-circuit protection – phase 1	Bit (0/1)->Word	Read/Write
ZW21	Short-circuit protection – phases 1 and 3	Bit (0/1)->Word	Read/Write
ZW22	Short-circuit protection – phases 1 and 2	Bit (0/1)->Word	Read/Write
ZW23	Short-circuit protection – phases 1, 2 and 3	Bit (0/1)->Word	Read/Write
ZW24	Independent overload protection – phase 0	Bit (0/1)->Word	Read/Write
ZW25	Independent overload protection – phase 3	Bit (0/1)->Word	Read/Write
ZW26	Independent overload protection – phase 2	Bit (0/1)->Word	Read/Write
ZW27	Independent overload protection – phases 2 and 3	Bit (0/1)->Word	Read/Write
ZW28	Independent overload protection – phase 1	Bit (0/1)->Word	Read/Write
ZW29	Independent overload protection – phases 1 and 3	Bit (0/1)->Word	Read/Write
ZW30	Independent overload protection – phases 1 and 2	Bit (0/1)->Word	Read/Write
ZW31	Independent overload protection – phases 1, 2 and	Bit (0/1)->Word	Read/Write
ZW32	Dependent overload protection – phase 0	Bit (0/1)->Word	Read/Write
ZW33	Dependent overload protection – phase 3	Bit (0/1)->Word	Read/Write
ZW34	Dependent overload protection – phase 2	Bit (0/1)->Word	Read/Write
ZW35	Dependent overload protection – phases 2 and 3	Bit (0/1)->Word	Read/Write
ZW36	Dependent overload protection – phase 1	Bit (0/1)->Word	Read/Write
ZW37	Dependent overload protection – phases 1 and 3	Bit (0/1)->Word	Read/Write
ZW38	Dependent overload protection – phases 1 and 2	Bit (0/1)->Word	Read/Write
ZW39	Dependent overload protection – phases 1, 2 and 3	Bit (0/1)->Word	Read/Write
ZW40	Ground-fault protection	Bit (0/1)->Word	Read/Write
ZW64	Undervoltage protection – phase 0	Bit (0/1)->Word	Read/Write
ZW65	Undervoltage protection – phase 3	Bit (0/1)->Word	Read/Write
ZW66	Undervoltage protection – phase 2	Bit (0/1)->Word	Read/Write
ZW67	Undervoltage protection – phases 2 and 3	Bit (0/1)->Word	Read/Write
ZW68	Undervoltage protection – phase 1	Bit (0/1)->Word	Read/Write
ZW69	Undervoltage protection – phases 1 and 3	Bit (0/1)->Word	Read/Write
ZW70	Undervoltage protection – phases 1 and 2	Bit (0/1)->Word	Read/Write
ZW71	Undervoltage protection – phases 1, 2 and 3	Bit (0/1)->Word	Read/Write

Table 46. Variable Storing States of Text Events.

Symb. Addr.	Variable in MUZRO	Type of Conversion	Allowed Operation
ZT1	Short-circuit PDZ (accelerated protections)	Bit (0/1)->Word	Read/Write
ZT4	Failure of shutdown	Bit (0/1)->Word	Read/Write
ZT5	Failure of switching on	Bit (0/1)->Word	Read/Write
ZT7	Opening of remote control	Bit (0/1)->Word	Read/Write
ZT8	Closure of remote control	Bit (0/1)->Word	Read/Write
ZT38	Technological protection 9	Bit (0/1)->Word	Read/Write
ZT39	Technological protection 10	Bit (0/1)->Word	Read/Write
ZT40	Technological protection 11	Bit (0/1)->Word	Read/Write
ZT41	Technological protection 12	Bit (0/1)->Word	Read/Write
ZT 48	Technological protection 13	Bit (0/1)->Word	Read/Write
ZT49	Technological protection 14	Bit (0/1)->Word	Read/Write
ZT50	Technological protection 15	Bit (0/1)->Word	Read/Write
ZT51	Technological protection 16	Bit (0/1)->Word	Read/Write
ZT52	Technological protection 17	Bit (0/1)->Word	Read/Write
ZT116	Bad clock setting	Bit (0/1)->Word	Read/Write
ZT117	Checksum error (MUZ failure)	Bit (0/1)->Word	Read/Write

Table 47. Control Variables.

Symb. Addr.	Variable in MUZRO	Type of Conversion	Allowed Operation
KS1	Quitting signals of protection activation (writing any	Word->Word	Writing
	and zeroing statuses of all variables **ZTi** and **ZWi**		
	in internal driver buffer		
SW1	Opening (0) and closure of breaker (1)	Word->Word	Writing
KZ1	Zeroing statuses of all variables **ZTi** and **ZWi**	Word->Word	Writing
	in internal driver buffer (writing any number)		

Meanings of Abbreviations

On	- protection switched on (1),
Off	- protection switched off (0),
Op	- protection causes opening of a breaker (1),
Sygn	- protection causes activation of a signalization (0),
Dep.	- protection is active when a breaker is closed (0),
Indep.	- protection is active independently.

Declaring the Permission for Control

Group handling the MUZ-RO devices has an effect on passing control permissions. It results form the fact that the ASMEN positions declaring control permissions refer to the individual logical channels. In the previous MUZ driver version (v 1.3.1.) the logical channel was assigned to the singular MUZ-RO device and it allowed passing permissions for each of MUZ-ROs individually. In the current group version of the driver the logical channel may group a few

MUZ-RO devices and permissions for control refer to all the MUZ-RO devices handled by the given logical channel.

Configuring the Driver

The driver configuration is performed by using a separate section named [MUZ]. By means of this section it is possible to declare:

- receiving timeout;
- log file and its size;
- data update;
- time update;
- log of telegrams;
- number of trials to do the command in case of transmission errors.

☑ ***RECV_TIMEOUT=number***

Meaning	- the item allows to determine a maximal waiting time between sending a query and receiving an answer (so called receiving timeout). The parameter value is passed for all handled MUZ-RO devices globally.
Default value	- 1000.
Defining	- manually.
Parameters:	
number	- timeout value in milliseconds.

☑ ***LOG_FILE=file_name, file_size***

Meaning	- allows to define a file to which all diagnostic driver messages and information about the content of telegrams sent/received by the driver will be written. If the item LOG_FILE does not defines the full path then the log file will be created in the current directory.
Default value	- log file is not created.
Defining	- manually.
Parameters:	
file_name	- log file name;
file_size	- log file size in MB;

☑ ***DATA_UPDATE=number***

Meaning	- the item allows to specify an acceptable time difference between the time stamp of the variable stored in the driver cache and the current system time. After exceeding the acceptable difference the driver performs reading the variable from the MUZ-RO device.
Default value	- by default the item assumes the value of 0.
Defining	- manual.
Parameters:	
number	- time value in seconds;

☑ ***TIME_UPDATE=number***

Meaning	- the item allows to specify the global period of updating the time between the **asix** system and the MUZ-RO device.
Default value	- by default the item assumes the value of 1.
Defining	- manual.
Parameters:	
number	- time value in minutes.

☑ ***LOG_OF_TELEGRAMS=YES/NO***

Meaning	- the item allows writing to the log file (declared with use of LOG_FILE item) the contents of telegrams transmitted during the data exchange between the **asix** system and MUZ-ROs; writing the telegrams content to the log file should be used only during starting the **asix** system.
Default value	- by default the file is not created.
Defining	- manual.

☑ ***NUMBER_OF_REPETITIONS=number***

Meaning	- the item allows to define maximal number of trials to do the command in case of transmission errors.
Default value	- by default the repetitions are not realized.
Defining	- manual.
Parameters:	
number	- number of repetitions in case of transmission error.

1.44. NetLink - Driver of MPI / Profibus Protocol for SIMATIC S7 by using NetLink Lite SYSTEME HELMHOLZ Module

Driver Use

The NetLink driver is used for data exchange between **asix** computers and SIMATIC S7 PLCs by using an MPI/Profibus bus and a NetLink Lite SYSTEME HELMHOLZ module.

Limitations of using NetLink Lite:
a) the module can handle maximally 2 clients at the same time;
b) during communication between a Netlink Lite module and PLCs (more than one) it is necessary to take into account a time needed for the module to switch over between PLCs (an exemplary configuration having been tested consisted of 3 PLCs, and the time of switching amounted to about 80 msec/PLC).

The NetLink driver needs ASMEN v. 4.6.8 or newer one.

Declaration of Transmission Channel

The syntax to declare the transmission channel operating with the NetLink driver is given below:

channel = NETLINK, IP, *port*, *address* [,*contr_var* [, *alarm_nr*] [, *error_signal*]]

where:

IP	- IP address assigned to the NetLink Lite module;
port	- port number (1099);
address	- PLC address on the MPI/Profibus Network;
contr_var	- variable name used for control of the PLC's RUN-STOP state;
alarm_nr	- alarm number generated when changing the PLC's RUN-STOP state; by default the alarm is not generated;
error_signal	- setting an error status for all the variables in a defined channel in case of switching the PLC to STOP; by default, an error status is set.

Addressing Process Variables

The rules for creating symbolic addresses of variables belonging to the channel that uses the NetLink driver are the same as for the SAPIS7 driver – see: SAPIS7 - Driver of SAPIS7 Protocol for SIMATIC S7 PLCs.

Driver Configuration

The NetLink protocol driver may be configured by use of the [NETLINK] section placed in the **asix** application INI file. All items in the section have the following format:

item_name = [*number*] [YES|NO]

☑ ***STATISTICS = yes/no***

Meaning	- the item allows to display (every 1 minute) information about number of transmission sessions that have been carried-out, average transmission time and number of transmission errors. The item was developed as a designer support on the stage of the **asix** system start-up.
Default value	- by default, the transmission statistics is not displayed.

☑ ***LOG_FILE=file_name***

Meaning	- the item allows to define a file to which all NetLink driver messages concerning operations performed by the driver are written. If the item does not define the full name, then the log file is created in the current directory.
Default value	- by default, the log file is not created.

☑ ***LOG_FILE_SIZE = number***

Meaning	- declares the log file size.
Default value	- 1MB.
Defining	- manual.
Parameters:	
number	- the log file size in MB;

☑ ***TELEGRAM_LOG = [YES|NO]***

Meaning	- declaration of writing the contents of telegrams sent and received by the NetLink driver within reading/writing process variables to the log file declared in the item LOG_FILE.
Default value	- NO.

Time Synchronization

See: *1.52. SAPIS7 - Driver of SAPIS7 Protocol for SIMATIC S7 PLCs*.

Signalization of the Controller’s STOP State

See: *1.52. SAPIS7 - Driver of SAPIS7 Protocol for SIMATIC S7 PLCs*.

1.45. NONE Driver

Driver Use

The NONE driver uses the NONE protocol internally served by ASMEN. That protocol doesn't realize a physical connection with the controller. It may by used for:

- application testing in simulation mode,
- data exchange between **asix** applications by means of process variables.

Driver Configuration

The full syntax of declaration of transmission channel which operates according to the NONE protocol is given below:

☑ ***WRITING_DATE_AND_STATUS=YES[NO]***

Meaning	- the item allows to determine a time period after exceeding of which the current time should be sent to remote devices.
Default value	- NO.

The item WRITNG_DATE_AND_STATUS causes that in the channel only writing with variable value supplemented with date and status is possible. At present this property is available only from the level of the **AsixConnect** function (*write2()*).

A status, time and variable value are a subject to the following rules:

- to the moment of the first writing, while reading, the status AVD_BAD is returned;
- while writing, the time given in writing parameters, new status and new variable value is written to the variable;
- the reading following the writing returns parameters of the lately performed writing (value, status and time);
- writing to the variable by using previous writing functions ends with an error code of 24;
- for needs of DEMO (program AS32_demo.exe) objects may performed writes to variables belonging to the NONE channel with the item WRITING_DATE_AND_STATUS=YES declared.

EXAMPLE

```
[ASMEN]
NONE1 = NONE

[NONE1]
WRITING_DATE_AND_STATUS=YES
```

1.46. OMRON - Driver of HOSTLINK Protocol

Driver Use

The OMRON driver protocol is used for data exchange with OMRON PLCs. The transmission is executed by means of serial interfaces HOSTLINK by using standard serial ports of an **asix** system computer.

The cooperation of **asix** with the controller by using the OMRON protocol does not require any controller's program adaptation. Before executing controls the driver switches the controller in MOTOROLA mode (if the controller is in the run mode). After control is ended, the controller is switched to mode, in which was operating before executing the control.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel operating according to the OMRON protocol is given below:

logical_name=OMRON,*type[,id],port,[baud,character,parity,stop]*

where:

type	- connection type - SLINK (single link), MLINK (multi link);
id	- controller identifier (unit number), it is used when the connection type was marked as MLINK (multi link);
port	- name of the serial port (COM1 or COM2);
baud	- transmission speed in baud;
character	- number of bits in a transmitted character;
parity	- parity check type (even,odd,none0);
stop	- number of stop bits.

The parameters *baud*, *character*, *parity*, *stop* are optional. In case of omitting them the following default values are assumed:

- transmission speed - 9600 Bd,
- number of bits in a character - 7,
- parity check type - parity check (even),
- number of stop bits - 2.

EXAMPLE

An exemplary declaration of transmission channel working according to the OMRON protocol:

```
CHAN1=OMRON,MLINK,0,COM1,9600,7,even,2
```

The transmission channel with the logical name CHAN1 has the following parameters defined:

- OMRON protocol using a serial interface working in MLINK (multi-link);
- identifier of the controller (unit number) 0;
- port COM1;
- transmission channel of 9600 Bd;
- transmitted character length - 7 bits;
- parity check;
- two stop bits.

Addressing the Process Variables

The syntax of symbolic address which is used for variables belonging to the OMRON driver channel is as follows:

variable_type variable_index

where:

variable_type	- string identifying the variable type in the OMRON protocol,
variable_index	- variable index within a given type.

The following symbols of process variables types are allowed (in paranthesis the ranges of variable types are given):

IR	- Internal Relay, (0 - 235, 300 - 511)
HR	- Holding Relay, (0 - 99)
AR	- Auxiliary Relay, (0 - 27)
LR	- Link Relay, (0 - 63)
DM	- Data Memory, (0 - 6143)

Driver Configuration

The HOSTLINK protocol driver may be configured by means of the **[OMRON]** section in the application INI file.

☑ ***LOG_FILE = file_name***

Meaning	- it allows to define a file to which all diagnostic messages of the driver and the information about the content of telegrams received by the driver will be written. If the item does not define the full path, then the log file will be created in the current directory. The log file should be used only while the **asix** start-up.
Default value	- log file is not created.
Defining	- manual

☑ ***LOG_FILE_SIZE = number***

Meaning	- it allows to determine a log file size in MB.
Default value	- 1MB
Defining	- manual

☑ ***LOG_OF_TELEGRAMS=[YES|NO]***

Meaning	- the item allows to write to the log file (declared by using an item LOG_FILE) the content of telegrams received by the driver. Writing the content of telegrams should be used only while the **asix** start-up.
Default value	- NO
Defining	- manual

☑ ***MAX_BUFFER_LEN = number***

Meaning	- maximal length of answer telegrams (counted in bytes). Maximal value is equal to 118.
Default value	- 118 bytes
Defining	- manual

EXAMPLES

IR22	- Internal Relay number 22
HR97	- Holding Relay number 97
DM6001	- Data Memory 6001

All process variable are treated as 16-bit numbers.

The OMRON driver is loaded as a DLL automatically.

1.47. OPC Driver

Driver Use

The OPC driver is used for data exchange between **asix** system computers and any industrial PLC or SCADA program that has an access to a data server compatible with the OPC 1.0 or OPC 2.05 specification. (OPC specification is available at http://www.opcfoundation.org).

The OPC driver handles operations of read/write from/into a controller of:
1. simple variables containing scalar values like:
   a. Byte - 8-bit unsigned number;
   b. Word - signed or unsigned word (16-bit number);
   c. Double Word - signed or unsigned double word (32-bit number);
   d. Single Precision - 32-bit real number;
2. single dimension table variables consisting of simple variables (*see: 1*);
3. text variables.

Changes in OPC Driver - Version 2.0

- Adding the OPC server service compatible with the OPC 2.05 specification.
- Adding the possibility of specifying the access path for variables in the transmission channel.
- Adding the possibility of blocking the operation of writing to the OPC server.
- Driver extension with read/write operation of text / table variables.
- Elimination of large OPC server loading reducing considerably its capacity in case when a variable archiving period was too long in relation to their sampling period.

Declaration of Transmission Channel

The definition of transmission channel using the OPC driver is as follows:

<Channel_name> = UniDriver, OPC, *<Options>*

The basic and the only obligatory OPC driver option is the OPC server identifier given in the following form:

ProgId = <OPC server identifier>

EXAMPLE

An exemplary declaration of the channel recalling to the OPC server registered under the *Matrikon.OPC.Simulation* identifier:

```
Matrikon = UniDriver, OPC, ProgId=Matrikon.OPC.Simulation
```

All the driver options are described in the table below.

Table 48. OPC Driver Options Used in the Channel Declaration.

Option	**Range and Value**	**Description**	**Default Value**
ProgId	<tekst>	OPC server identifier under which the server is registered in the Windows system.	Lack
ReadOnly	Yes | No	If the option has the value *Yes*, then all write operations to the OPC server are blocked.	No
ItemsAlwaysActive	Yes | No	The option should have the value *Yes* (with the exception of case when variables are not archived in the **asix** system and not used in scripts and the number of variables read at the same moment should be minimize because of a limited transmission band).	Yes
OPCVersion	0 | 1 | 2	If the option have the value 1 or 2, then communication with the OPC server is realized always according to the OPC specification of the version – properly 1.0 or 2.05. If the option have the value 0, then the driver tries to use both versions (beginning with 2.05).	0
ForcedReadRate	<number>	It determines the time (in milliseconds) after which an asynchronous read of all active measurements from the OPC server is performed.	No

An exemplary declaration of the channel using all options (it should be written in the application INI file in one line):

```
Matrikon = UniDriver, OPC, ProgId=Matrikon.OPC.Simulation, ReadOnly=yes,
ItemsAlwaysActive=yes, OPCVersion=2
```

Defining the Process Variables

Defining the variables in the **asix** system is described in *asix4, User Manual*, in chapter *Asmen – Communication Manager* (*see: Declaration of Process Variables*). When defining the variable (recalling to an OPC server), the variable identifier from the OPC server variable base should be declared as the variable address. The syntax of the identifier depends on the specific OPC server and is described in its documentation. If the identifier contains small letters, comma or white space, then the identifier (the whole one) should be put in quotation marks.

Type Matching

While initializing the variable in the OPC server, the OPC driver sends a requested type of the variable to it. This type is defined according to the conversion function that has been assigned to the variable in the **asix** VariableBase. If the variable type requested by the OPC driver differs from the type declared in the OPC variable base, then the OPC server usually accepts the requested type and performs the proper conversion during the data transmission. If the OPC server is not able to perform the proper conversion, then an error is notified at the moment of variable initialization. This fact is also signaled in the list of system messages (in *'Control Panel'*) as the message "*Incorrect variable type*". The variable the initialization of which finished with an error will have a bad status.

If the error connected with the variable type misfit in the **asix** system and the OPC server occurs, then the other analogical conversion function (but operating with the variable type handled by the OPC server) should be used. The variable types used by conversion functions are described in the **asix** system documentation, in chapter *ASMEN – Communication Manager/Conversion Functions* (the type of variables requested by the OPC server is named *type of PLC variable*).

The variable type misfit most often occurs when the conversion functions that operate on unsigned numbers are used. It is because some OPC servers don't handle such numbers at all. In such case, it is impossible to use the following functions: TIME, CIRCLE1, COUNTER, MASK, FACTOR, FACTOR_DW, NEGBIT, NEGBIT_DW, NOTHING, NOTHING_BYTE, NOTHING_DD, NOTHING_DW, ON/OFF, SHIFT_L, SHIFT_R, SLIODER, SLIDER1, SLIDER1_FP.

There are three conversion functions handling any type of values being received from the controller: GRADIENT, AVERAGE, TABLE. For these functions, the OPC driver always requests from the OPC server sending the variable values in the form of floating-point numbers – because the floating-point number is always these functions' output type.

Manager of Logical Channels for the OPC Driver

Manager of logical channels enables easy editing the definition of transmission channels using the OPC server through simple setting options in the dialog window, specially prepared for the OPC driver. The text edition of transmission channels using other drivers than the OPC driver is also possible.

The manager is placed in the *ChannelsManager.exe* file in the directory of the **asix** system (by default c:\asix). More detailed information you can find in Manager of the logical channels for the OPC driver.

Communication Testing

The OPC driver enters on the message list of *'Control Panel'* information on important events like driver loading, driver connection to the OPC server, variable initialization performance. The information on possible errors at the driver initialization and operation stage is also entered on the list.

Detailed information on errors and diagnostic information is placed in the OPC log file. The driver log file is named *UniDriver,<current_date>.log* and is placed in the **asix** system directory by default. There are two groups of options defining the kind of information written to the log file. The first group options are placed in the **[UniDriver]** section:

☑ ***LogPath =record_track***

Meaning	- the log file is placed in the defined directory.
Default value	- by default, the log file will be created in the asix directory.
Parameter:	
record_track	- path to the directory in which the log file will be created.

☑ ***ShowLogConsole = YES/NO***

Meaning	- if the option has the value YES, then the diagnostic window is displayed and all the information being written to the log appears in this window.
Default value	- NO.

☑ ***TracedNames = variable_list***

Meaning	- for each variable, which name is on the list, the information on its value, quality and stamp will be written to the log (during the variable processing).
Default value	- lack.
Parameter:	
variable_list	- list of the variable names in the asix system, delimited with commas.

The second group of options is placed in the section with the same name as the channel name to which they apply (see: the table below).

Table 49. Options Placed in the Section with the Same Name as the Channel Name for the OPC Driver.

Option	**Variable Range**	**Description**
LogOnDataChangeStat	Yes | No	The information on the number of variables sent each time by the OPC server, during the refreshing and processing of the data sent by the driver, is placed in the log.
LogOnDataChangeItemInfo	Yes | No	The names of variables sent each time by the OPC server during the refreshing are placed in the log.
LogAddItemSucceeded	Yes | No	The information on successful end of the operation of writing to the OPC server is placed in the log.
LogWriteSucceeded	Yes | No	The information on successful end of the operation of writing to the OPC server is placed in the log.
TracedNames	< list of the variable names in the **asix** system, delimited with commas >	For each variable, which name is on the list, the information on its value, quality and stamp will be written to the log (during the variable processing).

All the options concerning the communication test may be changed during the **asix** system operation. After the modification and writing the application initialization file on the disc, the new option values will be retrieved form the log by the OPC driver and will begin to have an effect on the range of information to be written to the log.

Channel Definition Updating for the OPC 1.0 Driver

The channel definition has the following form for the previous OPC driver version:

<Channel_name> = OPC, *<OPC server Progi>*

To convert the definition to the current obligatory form, one should change the text *"OPC,"* (with the comma) into *"UniDriver, OPC, ProgId =".*

EXAMPLE

Matrikon = OPC, Matrikon.OPC.Simulation

The current form:

Matrikon = UniDriver, OPC, ProgId = Matrikon.OPC.Simulation

1.48. PPI - Driver of PPI Protocol for SIMATIC S7 200 PLCs

Driver Use

The communication protocol PPI is used for data exchange between computers with the **asix** application and SIEMENS S7 200 PLCs.

Declaration of Transmission Channel

The syntax of declaration of transmission channel operating according to the PPI protocol has the following form:

logical_name=PPI,*address*,*COMn*

where:

n	- number of the serial port to which the controller is connected;
address	- address of the PLC.

Driver Configuration

Each defined channel may have its own section, the name of which is its logical name i.e. [logical_name]. In such section, parameters related only to a given station (logical channel) are placed. You should not place there parameters of serial transmission because they are related to a communication port, so to all controllers connected to a given port. The COMn port may have its own section named **[PPI:n]**. Values defined in such section become default values for all stations connected to a given port. In such section you should place parameters of serial transmission if they differ from the default values. If in the application INI file the section named **[PPI]** is placed, then values placed in such section become default values for all communication ports and stations supported by the driver. Values placed in the section of a given station ([logical_name]) have priority over values placed in the section of a given serial port, and the last ones have priority over the values placed in the section [PPI]. If a parameter is not found in any section, then it assumes its default value following the description below. In particular, the INI file may not include any sections parameterizing stations. Appropriate records defining a logical channel are only required in the section **[ASMEN]**.

Parameters of transmission via a serial interface can not to be placed in sections concerning individual stations.

☑ ***Baud =number***

☑ ***bps=number***

Meaning	- transmission speed; it is not placed in the parameter section of the logical channel.
Default value	- 9600.
Parameter:	
number	- number passed in Bd.

☑ ***parity =parity_parameter***

Meaning	- determines parity check type; it is not placed in the parameter section of the logical channel.	
Default value	- e.	
Parameter:		
parity_parameter		- of parity check type:
	n	- no parity bit,
	o	- odd parity check,
	e	- even parity check,
	m	- mark
	s	- space.

☑ ***word =number***

☑ ***word_length=number***

Meaning	- word length; it is not placed in the parameter section of the logical channel.
Default value	- 8.
Parameter:	
number	- number passed from the range of 5 - 8 bits.

☑ ***stop_bits =number***

Meaning	- number of stop bits; it is not placed in the parameter section of the logical channel.
Default value	- 1.
Parameter:	
number	- number of bits.

☑ ***retries =number***

☑ ***retry =number***

Meaning	- number of transmission repetitions in case of transmission errors.
Default value	- 4.
Parameter:	
number	- number of repetitions.

☑ ***time_out =number***

☑ ***timeout =number***

Meaning	- timeout for the station answer.
Default value	- 500.
Parameter:	
number	- time passed in milliseconds.

☑ ***Delay =number***

Meaning	- minimal time interval in milliseconds between transmissions of frames.
Default value	- 25.
Parameter:	
number	- time passed in milliseconds.

☑ ***AllErrors =yes/no***

Meaning	- if the parameter has a value of no, the information on *timeout* errors will appear in *'Control Panel'* only when the transmission missed in spite of attempts of its repetition. If it has a value of yes, the information on all errors is transmitted to *'Control Panel'*.
Default value	- no.

☑ ***AsComm =number***

Meaning	- determines whether the driver has to interoperate with the AsComm communication manager; it is not placed in the parameter section of the logical channel.
Default value	- no (from the version 1.1).

☑ ***Send_Frame =number***

Meaning	- maximal length of a sending frame.
Default value	- 117.
Parameter:	
number	- number passed in bytes from the range of 10-260.

☑ ***Receive_Frame =number***

Meaning	- maximal length of a receiving frame.
Default value	- 117.
Parameter:	
number	- number passed in bytes from the range of 10-260.

☑ ***Variables =number***

Meaning	- maximal number of variables transferred once.
Default value	- 8.

☑ ***Simulation =yes/no***

Meaning	- if *yes* is given, then data reading/writing from/to a PLC will be simulated.
Default value	- no.

☑ ***PCAdres =number***

Meaning	- address of a computer.
Default value	- no.
Parameter:	
number	- number passed in bytes from the range of 0-255.

EXAMPLE 1

```
[ASMEN]
.....
S7_212=PPI,5,COM2
....
```

In the example above, the station named S7_212 connected to the COM2 port is defined. The communication with the controller is operating on the basis of default parameters.

EXAMPLE 2

```
[ASMEN]
.....
S7_1=PPI,5,COM2
S7_2=PPI,6,COM2
S7_3=PPI,7,COM2
S7_4=PPI,8,COM3
S7_5=PPI,9,COM3
S7_6=PPI,10,COM4
....

[PPI]
;Default values for all stations
baud=9600

[PPI:3]
;Default values for all stations connected to the COM3 port
baud=19200

[S7_6]
delay = 15
```

In the example above stations with names from S7_1 to S7_6 are defined. The stations S7_1, S7_2 and S7_3 are connected to the COM2 port. The stations S7_4 and S7_5 are connected to the COM3 port. The station S7_6 is connected to the COM4 port. All serial ports except COM3 work with a speed of 9600 baud. The COM3 port works with a speed of 19200 baud. During data exchange with the station S7_6, a delay between transmissions will be reduced to 15 milliseconds.

NOTE *The PPI protocol driver may act together with the AsComm connections manager. In such case, the driver is registered as a client of the AsComm module with a name PPI:n, where n is a number of a serial interface through which the communication with the PLC is executed. In such case the section [PPI:n] may include driver parameters as well as parameters designated for the AsComm module.*

Defining the Process Variables

Measurement Data

The driver executes an access to the following variables.

Table 50. Variables Serviced by the PPI Driver.

Symbol	Data Length
Mn.m	BYTE
MBn	BYTE
MWn	WORD
MDn	DWORD
In.m	BYTE
Ibn	BYTE
Iwn	WORD
Idn	DWORD
Qn.m	BYTE
BN	BYTE
PWN	WORD
QDn	DWORD
Vn.m	BYTE
VBn	BYTE
VWn	WORD
VDn	DWORD
Sn.m	BYTE
SBn	BYTE
SWn	WORD
SDn	DWORD
SMn.m	BYTE
SMBn	BYTE
SMWn	WORD
SMDn	DWORD
AIWn	WORD
AQWn	WORD
HCn	DWORD
Cn	WORD
Cn.m	BYTE
Tn	WORD
Tn.m	BYTE
RUN	BYTE

Meaning of symbols placed in the left column (except RUN) is described in the documentation of S7 controllers.

The variable RUN assumes a value of 1 if the controller is in the state RUN, and of 0 otherwise. Writing to the variable RUN causes an activation of the controller. Writing the value of 0 to the variable RUN causes a transition of the controller to the STOP mode. The change of the controller state is possible only at a suitable setting of switches on the controller.

The variables Cn and Tn enable to access to an actual value of counters and timers. The variables Cn.m and Tn enable access to the state (1 or 0) of counters and timers. The value m.may be any number of the range 0 to 7.

The present driver version does not allow to write to the variables Q, AQW, AIW. Writing to other variables is limited by the controller (Cm.n, Tm.n).

Access to Pseudo-Variables

The PPI protocol driver enables access to pseudo-variables. The access to pseudo-variables does not cause a physical transmission through a serial interface. Values of pseudo-variables are related with an actual state of a connection with the controller.

Table 51. Pseudo-Variables Serviced by the PPI Driver.

Symbol	Meaning	Length
SBS	number of sent bytes	DWORD
SBR	number of received bytes	DWORD
SFS	number of sent frames	DWORD
SFR	number of received frames	DWORD
SPE	number of parity errors	DWORD
SFE	number of frame errors	DWORD
SOE	number of overrun errors	DWORD
SLE	number of line errors (sum of parity, frame, overrun and other errors)	DWORD
STE	number of timeout errors	DWORD
SPRE	number of protocol errors	DWORD
SCE	number of checksum errors	DWORD
SFC	number of unsuccessful connections (by means of AsComm module)	DWORD
SBC	number of broken connections (established by AsComm module)	DWORD
SLGE	number of logical errors (no data in the controller, faulty address etc.).	DWORD
ERR	sum of all errors (SLE, STE, SPRE, SCE, SFC, SBC and SLGE). Writing of any value to ERR variable causes zeroing of variables SBS, SBR, SFS, SFR, SPE, SFE, SOE, SLE, STE, SPRE, SCE, SFC, SBC and SLGE.	DWORD
TSBS	number of sent bytes (from the beginning of driver operation)	DWORD
TSBR	number of received bytes (from the beginning of driver operation)	DWORD
TSFS	number of sent frames (from the beginning of driver operation)	DWORD
TSFR	number of received frames (from the beginning of driver operation)	DWORD
TSPE	number of parity errors (from the beginning of driver operation)	DWORD
TSFE	number of frame errors (from the beginning of driver operation)	DWORD
TSOE	number of overrun errors (from the beginning of driver operation)	DWORD
TSLE	number of line errors (sum of parity, frame, overrun and other errors) (from the beginning of driver operation)	DWORD
TSTE	number of timeout errors (from the beginning of driver operation)	DWORD
TSPRE	number of protocol errors (from the beginning of driver operation)	DWORD
TSCE	number of checksum errors (from the beginning of driver operation)	DWORD
TSFC	number of unsuccessful connections (by means of AsComm module) (from the beginning of driver operation)	DWORD
TSBC	number of broken connections (established by AsComm module) (from the beginning of driver operation)	DWORD
TSLGE	number of logical errors (no data in the controller, faulty address etc.). (from the beginning of driver operation)	DWORD
TERR	sum of errors determined by variables TSLE, TSTE, TSPRE, TSCE, TSFC, TSBC and TSLGE.	DWORD
ONLINE	assume a value of 1 if the last attempt to send any frame ended successfully (i.e. an acknowledge from the controller was received) and 0 otherwise.	BYTE

1.49. CtProtherm300 - Driver of PROTHERM 300 DIFF PLC Protocol

Driver Use

The CtProtherm300 driver is used for data exchange between **asix** system computers and Protherm 300 DIFF PLC of Process-Electronic GmbH. The data are transferred over the RS-422 serial link.

Declaration of Transmission Channel

The CtProtherm300 driver is loaded by the universal **asix** system driver – UNIDRIVER. The declaration of transmission channel using the CtProtherm300 driver is as follows:

Channel=UNIDRIVER, CtProtherm300, Port=*port_number* [;RecvTimeout=*ms_number*]

where:

UNIDRIVER	- universal **asix** system driver;
CtProtherm300	- name of the driver for communication with the Protherm PLC;
port_number	- number of the serial link (1 is passed for COM1, 2 for COM2, end so on);
ms_number	- timeout of waiting for the controller response (in milliseconds); it is passed 1000 milliseconds by default.

By default, the following transmission parameters are passed:

- transmission speed 9600 Bd;
- number of bytes in a character;
- parity check –EVEN;
- number of stop bytes – 1.

EXAMPLE

An exemplary declaration of the channel using the CtProtherm300 driver on the serial port COM2 with receiving timeout that equals 2000 ms is as follows:

```
PLC1 = UNIDRIVER, CTPROTHERM300, Port=2; RecvTimeout=2000
```

Addressing the Process Variables

The syntax of the variable address is as follows:

V.*group.unit.index*

where:

group	- number of the group to which the controller is assigned (a number form 0 to 9);
unit	- number assigned to the controller within the group (a number form 0 to 9);
index	- number of the variable in the controller, compatible with the specification *'Variablen-Liste fuer PT300'* (in P*rotherm300/ Protherm300 Kommunikation* documentation).

The raw variables may have one of the following type:

- byte;
- byte table;
- float.

The type of raw variable is defined in the specification *'Variablen-Liste fuer PT300'*, in *Protherm300/Protherm300 Kommunikation* documentation with the symbol R411.0, published by Process-Electronic.

EXAMPLE

Examples of variable declaration.

```
JJ_00, value CO (FLOAT),V.1.2.6,          PLC1, 1, 1, NOTHING_FP
JJ_01, status STAT3 (BAJT),V.1.2.18,      PLC1, 1, 1, NOTHING_BYTE
JJ_02, table of 4 statuses XRELi (BAJT),V.1.2.14,PLC1, 4, 1, NOTHING_BYTE
```

Stamp

The variables read from the Protherm300 system are stamped with a local PC time.

Configuration

The driver configuration is performed by using the separate section named **[CTPROTHERM300]**. By means of this section it is possible to declare:

- log file and its size,
- log of telegrams.

LOG_FILE=file_name

Meaning	- the item allows to define a file to which all the diagnostic messages of the driver will be written. If the item LOG_FILE doesn't define a full path, the log file will be created in the current directory. The log file should be used only during the **asix** system start-up.
Default value	- by default, the log file is not created.

Parameter:

file_name - name of the log file.

LOG_FILE_SIZE=number

Meaning	- allows to define the size of the log file in MB.

Default value	- by default, the item assumes that the log file has a size of 1 MB.
Parameter:	
number	- size of the log file in MB.

☑ ***LOG_OF_TELEGRAMS=YES/NO***

Meaning	- the item allows writing to the log file (declared with use of the LOG_FILE item) the contents of telegrams transmitted during the data exchange between the **asix** system and Protherm 300 DIFF PLC of Process-Electronic GmbH company; writing the telegrams content to the log file should be used only during the **asix** system start-up
Default value	- NO.

1.50. PROTRONICPS - Driver of PROTRONICPS Regulator Protocol

Driver Use

The PROTRONICPS driver is used for data exchange between PROTRONIC PS regulators of Hartmann & Braun and an **asix** system computer. The communication is performed by means of serial interfaces in the RS422 standard.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel operating according to the PROTRONICPS protocol is given below:

logical_channel_name=PROTRONICPS, *id, port, baud [,character, parity, stop]*

where:

PROTRONICPS	- driver name,
id	- device identifier (regulator no. in the network),
port	- name of the serial port,
baud	- transmission speed,

optional parameters:

character	- number of bits in a character,
parity	- parity check type,
stop	- number of stop bits.

Default values:

- 8 bits in a character,
- even parity check (EVEN),
- number of stop bits 1.

The PROTRONICPS driver is loaded as a DLL automatically.

Addressing the Process Variables

The syntax of symbolic address which is used for variables belonging to the PROTRONICPS driver channel is as follows:

<type><index>

where:

type	- variable type; allowable types are:
	STAT - controller status,
	F - status of controller errors,
	WA - analog value,
	BV - binary value,
	Y - controller output;
index	- index within the type (used only for WA, BV and Y):
	WA 0 – 255
	BV 0 – 255
	Y 0 – 10

Variables of STAT and F type may be only read.
Variables of Y type may be only written.
Variables of WA and BV type may be read and written.
Raw values of STAT, F, WA type variables are of WORD type.
Raw values of BV type variables are of BYTE type.
Raw values of Y type variables are of SIGNED SHORT type.

EXAMPLE

An exemplary declaration of variables:

```
X1,  Controller status,STAT,        CHAN1,  1,  1,  NOTHING
X2,  Status of errors,F,            CHAN1,  1,  1,  NOTHING
X3,  Bit BV1,BV1,                   CHAN1,  1,  1,  NOTHING_BYTE
X4,  Analog WA3,WA3,                CHAN1,  1,  1,  NOTHING
X5,  Controller output no. 0,Y0,    CHAN1,  1,  1,  NOTHING
X6,  Controller output no. 1,Y1,    CHAN1,  1,  1,  NOTHING
```

Driver Configuration

The PROTRONICPS protocol driver may be configured by use of the **[PROTRONICPS]** section in the application INI file. Individual parameters are transferred in separated items of the section. Each item has the following syntax:

item_name=[number [,number]] [YES|NO]

☑ ***LOG_FILE=file_name***

Meaning	- the item allows to define a file to which all diagnostic messages of the PROTRONICPS driver and information about contents of telegrams received by the driver are written. If the item does not define the full path, then the log file is created in the current directory. The log file should be used only while the **asix** start-up.
Default value	- by default, the log file is not created.

☑ ***LOG_OF_TELEGRAMS=YES|NO***

Meaning	- the item allows to write to the log file (declared by LOG_FILE) the contents of telegrams sent within the communication with the PROTRONIC PS controller. Writing the contents of telegrams to the log file should be used only while the **asix** start-up.
Default value	- by default, telegrams are not written.

☑ ***NUMBER_OF_REPETITIONS=YES|NO***

Meaning	- the item allows to determine a number of repetitions in case of occurrence of transmission error.
Default value	- by default, telegrams are not written.

1.51. S700 - Driver S700 of MAIHAK Analyzer Protocol

Driver Use

The S700 driver is used for data exchange between Maihak S700 gas analyzers and an **asix** system computer by use of the AK protocol. The communication is performed via standard serial ports of an **asix** computer and the serial interface no. 1 of the analyzer.

NOTE Settings *Serial interface # 1*of the analyzer must have the following values:

1 / RTS/CTS protocol - without RTS/CTS protocol,
2 / XON/XOFF protocol - without XON/XOFF protocol.

Settings *Communication # 1* of the analyzer must have the following values:
1 / AK-ID active - ON (1).

Declaration of Transmission Channel

The full syntax of declaration of transmission channel operating according to the S700 protocol is given below:

logical_channel_name=S700, *id, port [, baud ,character, parity, stop]*

where:

S700	- driver name,
id	- analyzer identifier (number AK-ID),
port	- port name: COM1, COM2 etc.,

optional parameters:

baud	- transmission speed,
character	- number of bits in a character,
parity	- parity check type,
stop	- number of stop bits.

If optional parameters are not given, then default values are as follows:

- transmission speed of 9600 baud,

- 8 bits in a character,
- no parity check (NONE),
- number of stop bits 1.

EXAMPLE

The declaration of the logical channel named CHAN1 operating according to the S700 protocol and exchanging data with the analyzer no. 1 via the COM2 port is is follows:

```
CHAN1=S700, 1, COM2
```

The S700 driver is loaded as a DLL automatically.

Addressing the Process Variables

The syntax of symbolic address which is used for variables belonging to the S700 driver channel is as follows:

<*type*>[<*index*>][.<*element*>]

where:

type	- variable type,
index	- index within the type (for some types of variables),
element	- element within the index (for some types of variables).

Types which use no index (in parentheses a type of raw variable value is given):

CZK	- time of calibration measure (WORD),
CZO	- delay time (WORD),
IDA	- analyzer identifier (up to 40 characters) (BYTE),
MNU	- menu language identifier (BYTE),
NRS	- analyzer serial number (BYTE),
PGB	- measure (introducing) of tested gas to the analyzer (WORD),
SPKK	- state of a calibration tray pump (WORD).

Types which need to give an index (in parentheses a type of raw variable value is given):

DWK	- sensibility drift after calibration (FLOAT),
DZK	- zero-point drift of measure substance after calibration (FLOAT),
NSKK	- rated value of measured substance in calibration tray (FLOAT),
PGKW	- measuring (introducing) of standard calibration gas in the analyzer (WORD),
PGKZ	- measuring (introducing) of zero calibration gas to the analyzer (WORD),
SKT	- compensation of measured substance temperature (WORD),
SPKW	- status of a standard calibration gas pump (WORD),
SPKZ	- status of a zero calibration gas pump (WORD),
SPT	- name of measure substance (BYTE),
SPW	- current values of measured substances (FLOAT),
SPZ	- end value of measuring range of measured substance (FLOAT),
STA	- actual analyzer state (WORD).

Types which need to give an index and an element (in parentheses a type of raw variable value is given):

NSPW	- rated value of measured substance in standard calibration gas (FLOAT),
NSPZ	- rated value of measured substance in zero calibration gas (FLOAT).

Types only for reading :

DWK	- sensibility drift after calibration,
DZK	- zero-point drift of calibration substance after calibration,
MNU	- menu language identifier,
NRS	- analyzer serial number,
NSKK	- rated value of measure substance in calibration tray,
SPKK	- status of a calibration tray pump,
SPT	- name of measured substance,
SPW	- current values of measured substance,
SPZ	- limit range value of measured substance,
STA	- current analyzer status.

Types only for writing:

PGB	- measuring (introducing) of tested gas in the analyzer,
PGKW	- measuring (introducing) of standard calibration gas to the analyzer,
PGKZ	- measuring (introducing) of zero calibration gas to the analyzer.

Types for reading and writing:

CZO	- delay time,
CZK	- time of calibration measurement,
IDA	- identifier of the analyzer,
NSPW	- rated value of the measured substance in zero calibration gas,
NSPZ	- rated value of measured substance in zero calibration gas,
SKT	- compensation of measured substance temperature,
SPKW	- status of a standard calibration gas pump,
SPKZ	- status of a zero calibration gas pump.

EXAMPLE

Exemplary declarations of variables.

```
# names of types (SPT) and ranges (SPZ) of measured substances – with an
index
X1,  SPT1,  CHAN1, 10, 1, NOTHING_TEXT
X2,  SPZ1,  CHAN1, 1,  1, NOTHING_FP
X3,  SPT2,  CHAN1, 10, 1, NOTHING_TEXT
X4,  SPZ2,  CHAN1, 1,  1, NOTHING_FP
X5,  SPT3,  CHAN1, 10, 1, NOTHING_TEXT
X6,  SPZ3,  CHAN1, 1,  1, NOTHING_FP
X7,  SPT4,  CHAN1, 10, 1, NOTHING_TEXT
X8,  SPZ4,  CHAN1, 1,  1, NOTHING_FP
X9,  SPT5,  CHAN1, 10, 1, NOTHING_TEXT
X10, SPZ5,  CHAN1, 1,  1, NOTHING_FP

# state bytes (STA) of analyzer with an index
X11, STA1, CHAN1, 1, 1, NOTHING
X12, STA2, CHAN1, 1, 1, NOTHING
```

```
X13,  STA3,  CHAN1, 1, 1, NOTHING
X14,  STA4,  CHAN1, 1, 1, NOTHING
X15,  STA5,  CHAN1, 1, 1, NOTHING
X16,  STA6,  CHAN1, 1, 1, NOTHING
X17,  STA7,  CHAN1, 1, 1, NOTHING
X18,  STA8,  CHAN1, 1, 1, NOTHING

# actual values (SPW) of measured substances – with an index
X21,  SPW1,  CHAN1, 1, 1, NOTHING_FP
X22,  SPW2,  CHAN1, 1, 1, NOTHING_FP
X23,  SPW3,  CHAN1, 1, 1, NOTHING_FP
X24,  SPW4,  CHAN1, 1, 1, NOTHING_FP
X25,  SPW5,  CHAN1, 1, 1, NOTHING_FP

# time pauses (CZO and CZK) – without an index
X31,  CZO,    CHAN1,  1, 1, NOTHING
X32,  CZK,     CHAN1,  1, 1, NOTHING

# results after calibration: zero drift (DZK), sensibility drift (DWK) – without an
index
X42,  DZK1,   CHAN1,  1, 1, NOTHING_FP
X43,  DWK1,  CHAN1,  1, 1, NOTHING_FP

X44,  DZK2,   CHAN1,  1, 1, NOTHING_FP
X45,  DWK2,  CHAN1,  1, 1, NOTHING_FP

# status of temperature compensation (SKT) – with an index
X51,  SKT1,  CHAN1, 1, 1, NOTHING
X52,  SKT2,  CHAN1, 1, 1, NOTHING
X53,  SKT3,  CHAN1, 1, 1, NOTHING
X54,  SKT4,  CHAN1, 1, 1, NOTHING
X55,  SKT5,  CHAN1, 1, 1, NOTHING

# identifier of analyzer (IDA) – without an index
X61,  IDA,   CHAN1, 42, 1, NOTHING_TEXT

#serial number of analyzer (IDA)  - without an indexes
X62,  NRS,   CHAN1, 20, 1, NOTHING_TEXT

# menu language of analyzer (IDA)  - without an index
X63,  MNU,   CHAN1, 1, 1, NOTHING_BYTE

# state of zero calibration gas pump (1 and 2) (SPKZ) – with an index
X70,  SPKZ1,   CHAN1, 1, 1, NOTHING
X80,  SPKZ2,   CHAN1, 1, 1, NOTHING

# nominal values of zero calibration gases (1 and 2) (NSPZ) – with an index
and element
X71,  NSPZ1.1, CHAN1, 1, 1, NOTHING_FP
X72,  NSPZ1.2, CHAN1, 1, 1, NOTHING_FP
X73,  NSPZ1.3, CHAN1, 1, 1, NOTHING_FP
X74,  NSPZ1.4, CHAN1, 1, 1, NOTHING_FP
X75,  NSPZ1.5, CHAN1, 1, 1, NOTHING_FP

X81,  NSPZ2.1, CHAN1, 1, 1, NOTHING_FP
X82,  NSPZ2.2, CHAN1, 1, 1, NOTHING_FP
X83,  NSPZ2.3, CHAN1, 1, 1, NOTHING_FP
X84,  NSPZ2.4, CHAN1, 1, 1, NOTHING_FP
```

```
X85,  NSPZ2.5, CHAN1, 1, 1, NOTHING_FP

# status of standard calibration gas pump (3 - 6) (SPKW) – with an index
X90,   SPKW3,   CHAN1, 1, 1, NOTHING
X100, SPKW4,   CHAN1, 1, 1, NOTHING
X110, SPKW5,   CHAN1, 1, 1, NOTHING
X120, SPKW6,   CHAN1, 1, 1, NOTHING

# rated values of standard calibration gases (3 - 6) (NSPW) – with an index
and element
X91,  NSPW3.1, CHAN1, 1, 1, NOTHING_FP
X92,  NSPW3.2, CHAN1, 1, 1, NOTHING_FP
X93,  NSPW3.3, CHAN1, 1, 1, NOTHING_FP
X94,  NSPW3.4, CHAN1, 1, 1, NOTHING_FP
X95,  NSPW3.5, CHAN1, 1, 1, NOTHING_FP

X101,  NSPW4.1, CHAN1, 1, 1, NOTHING_FP
X102,  NSPW4.2, CHAN1, 1, 1, NOTHING_FP
X103,  NSPW4.3, CHAN1, 1, 1, NOTHING_FP
X104,  NSPW4.4, CHAN1, 1, 1, NOTHING_FP
X105,  NSPW4.5, CHAN1, 1, 1, NOTHING_FP

X111,  NSPW5.1, CHAN1, 1, 1, NOTHING_FP
X112,  NSPW5.2, CHAN1, 1, 1, NOTHING_FP
X113,  NSPW5.3, CHAN1, 1, 1, NOTHING_FP
X114,  NSPW5.4, CHAN1, 1, 1, NOTHING_FP
X115,  NSPW5.5, CHAN1, 1, 1, NOTHING_FP

X121,  NSPW6.1, CHAN1, 1, 1, NOTHING_FP
X122,  NSPW6.2, CHAN1, 1, 1, NOTHING_FP
X123,  NSPW6.3, CHAN1, 1, 1, NOTHING_FP
X124,  NSPW6.4, CHAN1, 1, 1, NOTHING_FP
X125,  NSPW6.5, CHAN1, 1, 1, NOTHING_FP

# settings of calibration tray (SPKK) - pump, (NSKK) - rated
X130,  SPKK,    CHAN1, 1, 1, NOTHING
X131,  NSKK1,   CHAN1, 1, 1, NOTHING_FP
X132,  NSKK2,   CHAN1, 1, 1, NOTHING_FP
X133,  NSKK3,   CHAN1, 1, 1, NOTHING_FP
X134,  NSKK4,   CHAN1, 1, 1, NOTHING_FP
X135,  NSKK5,   CHAN1, 1, 1, NOTHING_FP

# start of zero calibration gas (1 and 2)
X201,  PGKZ1,   CHAN1, 1, 1, NOTHING
X202,  PGKZ2,   CHAN1, 1, 1, NOTHING

# start of standard calibration gas measuring (3 – 6)
X203,  PGKW3,   CHAN1, 1, 1, NOTHING
X204,  PGKW4,   CHAN1, 1, 1, NOTHING
X205,  PGKW5,   CHAN1, 1, 1, NOTHING
X206,  PGKW6,   CHAN1, 1, 1, NOTHING

# start of tested gas measuring
X207,  PGB,     CHAN1, 1, 1, NOTHING
```

Driver Configuration

The S700 protocol driver may be configured by use of the section **[S700]** placed in the application INI file. Individual parameters are transferred in separate items of the section. Each item has the following syntax:

item_name=[number [,number]] [YES] [NO]

☑ ***LOG_FILE=file_name***

Meaning	- the item allows to define a file to which all diagnostic messages of the S700 driver and information about contents of telegrams received by the driver are written. If the item does not define the full path, then the log file is created in the current directory. The log file should be used only while the **asix** start-up.
Default value	- by default, the log file is not created.

☑ ***LOG_OF_TELEGRAMS=YES|NO***

Meaning	- the item allows to write to the log file (declared by use of LOG_FILE) the contents of telegrams transferred within the communication with a S700 analyzer. Writing the contents of telegrams to the log file should be used only while the **asix** start-up.
Default value	- by default, telegrams are not written.

☑ ***LOG_FILE_SIZE=number***

Meaning	- the item allows to specify the log file size in MB.
Default value	- by default, the item assumes that the log file has a size of 1 MB.

☑ ***IGNORE_STATUS_CHARACTER=YES|NO***

Meaning	- in each answer from S700 an internal status byte of the analyzer is transferred. The content of this byte decides about the status of data which are transferred in a given answer from the analyzer. If the byte has a value of 0, then the variables receive a correct status, otherwise they receive an error status. The use of the item IGNORE_STATUS_CHARACTER with a value of YES causes that a correct status is assigned to the variables, irrespectively of the status byte content.
Default value	- by default, the item has a value of NO.

☑ ***RECV_TIMEOUT=station_no,number***

Meaning	- the item RECV_TIMEOUT allows to specify a waiting time for arriving the first character of an answer sent from a specified analyzer. After passage of this time it is assumed that a given analyzer does not work correctly and the transmission session ends with an error.

Default value	- by default, it is assumed that the maximal waiting time for the first character of an answer is equal to 1000 milliseconds.

Parameter:

station_no	- number AK-ID of the analyzer,
number	- time in milliseconds (from 100 to 5000).

☑ ***CHAR_TIMEOUT=station_no,number***

Meaning	- the item allows to determine a maximal time between successive characters of answer from a given analyzer. After passage of this time it is assumed that a given analyzer does not work correctly and transmission session ends with an error.
Default value	- by default, it is assumed that the time between successive characters is equal to 50 millisecond.

Parameter:

station_no	- number AK-ID of analyzer,
number	- time in milliseconds (from 10 to 300).

1.52. SAPIS7 - Driver of SAPIS7 Protocol for SIMATIC S7 PLCs

Driver Use

The SAPIS7 driver is used for data exchange with SIMATIC S7 PLCs by means of the MPI interface or a PROFIBUS bus communication processor. In an **asix** system computer the SIEMENS CP5611, CP5412(A2) or CP5613 card is used. The data exchange with use of the SAPIS7 protocol is based on so called S7 functions.

Operation of the **asix** system with the SIMATIC S7 PLC by use of the SAPIS7 protocol does not require any controller's program adaptation for data exchange.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel operating according to the SAPIS7 protocol is given below:

> *channel*=SAPIS7, *device, connection [,control_var [, alarm_nr] [, error_signal]]*

where:

device	- virtual device name (*VFD*),
connection	- connection name,
control_var	- name of a variable used to control the RUN-STOP state of the PLC,
alarm_nr	- number of the alarm generated when the RUN-STOP state of the PLC changes; by default, alarms are not generated;
error_signal	- setting an error status for all variables of a given channel in case of the PLC switching over into the STOP state; by default, an error status is set.

Names *VFD* and *connection name* must be compatible to parameters declared by use of the configuration program COML S7 supplied with the communication processor board.

EXAMPLE

An exemplary item declaring use of transmission channel operating according to the SAPIS7 protocol is given below:

CHAN1=SAPIS7, VFD1, S7_connection1

The logical channel named CHAN1 has defined the following parameters:

- SAPIS7 protocol,
- virtual device name (*VFD*) - VFD1,
- connection name - S7_connection1.

Rules of creating the symbolic addresses of variables belonging to the transmission channel using the SAPIS7 protocol are the same ones as in case of a channel using the AS512 protocol.

The set of process variable types used in the SAPIS7 protocol was extended with the following elements in relation to the set offered by the AS512 protocol:

EDI	- 16-byte words in INTEL convention,
ER	- content of data blocks, treated as floating-point numbers,
EB	- content of data blocks, treated as bytes.

The SAPIS7 driver is loaded as a DLL automatically.

Driver Configuration

The SAPIS7 protocol driver may be configured by use of the **[SAPIS7]** section placed in the application INI file. All items in the section have the following format:

item_name = [number] [YES|NO]

☑ ***WITHOUT_MULTIPLE_READ = YES/NO***

Meaning	- it allows to block the mode of multiple read creation and enables driver operation in the mode of single read.
Default value	- *No*, in the current version the *s7_multiple_read_req* function is used, by default (it allows to build a multiple read), for maximal use of the length of the telegram buffer sent between PC and S7.
Parameter:	
number	- length of telegrams passed in bytes.

☑ ***MAX_BUFFER_LEN = number***

Meaning	- the length of telegrams accepted by an MPI interface depends on the CPU type of S7 controllers and on a program executed by them.
Default value	- the default, length of a telegram is equal to 220 bytes.
Parameter:	
number	- length of telegrams passed in bytes.

☑ ***STATISTICS = yes/no***

Meaning	- the item allows to display (every 1 minute) information about number of transmission sessions that have been carried-out, average transmission time and number of transmission errors. The item was developed as a designer support on the stage of the **asix** system start-up.
Default value	- by default, the transmission statistics is not displayed.

☑ ***CONSOLE=YES|NO***

Meaning	- the item allows to create a console window where SAPIS7 driver messages concerning operations executed by the driver are displayed currently.
Default value	- by default, any console window is not created.

☑ ***LOG_FILE=file_name***

Meaning	- the item allows to define a file to which all SAPIS7 driver messages concerning operations performed by the driver are written. If the item does not define the full name, then the log file is created in the current directory.
Default value	- by default, the log file is not created.

Time Synchronization

By using the SAPIS7 driver it is possible to synchronize an **asix** station time with a controller time by the following item.

☑ ***TIME_SYNCHRONIZATION=channel_name,variable_name***

Parameters:

CHANNEL	- name of the ASMEN channel using the SAPIS7 protocol;
VARIABLE	- name of the ASMEN variable belonging to the channel CHANNEL and used for time synchronization.

The time synchronization algorithm consists in cyclic writing a frame containing an actual **asix** time to the S7. The frame is written according to the VARIABLE variable address and a refreshing frequency assigned to the variable VARIABLE.

The variable VARIABLE must be an array with a size of min. 10 bytes (the size of a time frame).

The time frame format (all the data in BCD format):

Byte	Contents	Range
0	Year	1990 to 2089 (only two last digits)
1	Month	01 to 12
2	Day	1 to 31
3	Hour	0 to 23
4	Minute	0 to 59
5	Second	0 to 59
6	millisecond	00 to 99 two most significant tetrads of milliseconds

7	millisecond	0 to 9	the least significant tetrad of millisecond written on the older tetrad of the byte no. 7
7	week day	1 to 7	written on the lower tetrad of the byte no. 7 (Sunday=1)
8	Mark of new time	1	
9	Irrelevant	always 0	

EXAMPLE

The time synchronization in the channel CHAN1 is executed by means of the variable X1. The time frame is written every 30 seconds to the data block DB100 from the byte 0 to the byte 9 inclusive:

```
; declaration of the variable X1
X1, EB100.0, CHAN1, 10, 30, NOTHING_BYTE

; declaration of time synchronization bye means of the variable X1
[ASMEN]
CHAN1 = SAPIS7,VFD2,conn_4
TIME_SYNCHRONIZATION = CHAN1, X1
```

Signalization of Controller STOP State

In the configuration with the S7 controller where the communication processor is independent of the central processor, in order to signal a controller STOP state correctly you should:

- declare in the controller a 1-byte control variable the value of which is changed during the processor work;
- declare the same variable as an ASMEN variable;
- declare (in the ASMEN section beside the logical channel declaration) a checking of operation by verification of control variable changes.

EXAMPLE

The variable named S7_CONN_0 is declared in the channel SINEC1 as a control variable:

```
S7_CONN_0 control byte RUN PLC 1, EM0, SINEC1, 1, 1, NOTHING_BYTE
```

The declaration of the channel SINEC1 in the ASMEN section:

```
    SINEC1=SAPIS7,VFD1,SMOLA_OP1,S7_CONN_0,1,yes
```

Control parameters:

S7_CONN_0	- control variable name;
,1	- generated alarm number;
,yes	- determines whether an alarm of communication breakdown has to be generated.

Additional items:

☑ ***NUMBER_OF_CHECK_READINGS=<number>***

Meaning	- the item specifying a minimal number of successive readings of control variable (of a constant value) which cause a signalization of the controller STOP status.
Default value	- by default, the item assumes a value of 3.

☑ ***TELEGRAM_LOG = [YES|NO]***

Meaning	- declaration of writing the contents of telegrams sent and received by the SAPIS7 driver within reading/writing process variables to the log file declared in the item LOG_FILE.
Default value	- NO.

☑ ***SERIALISING = [YES|NO]***

Meaning	- declaration of servicing the transmission from the S7 by transferring single YES or many NO queries.
Default value	- YES.

1.53. S-BUS - Driver of S-BUS Protocol for SAIA-Burgess Electronics PLCs

Driver Use

The S-BUS driver is used for data exchange between PCD PLCs of SAIA-Burgess Electronics and an **asix** system computer.

The S-BUS protocol is compatible to the specification *"SAIA S-Bus for the PCD family"*, *Edition 26/739 E2-05.96, developed by SAIA-Burgess Electronics*.

For purposes of communication with the **asix** system, the following interfaces of PCD PLCs may be used:

- the PGU interface (RS-232C);
- additional communication interfaces, the number and type of which depend on the controller type and configuration. These interfaces enable data transmission in one of the standards given below:
  RS-232C,
  RS-422,
  RS-485,
  current loop of 20 mA.

NOTE *For data exchange you should use a cable made according to the specification PCD8.K111.*

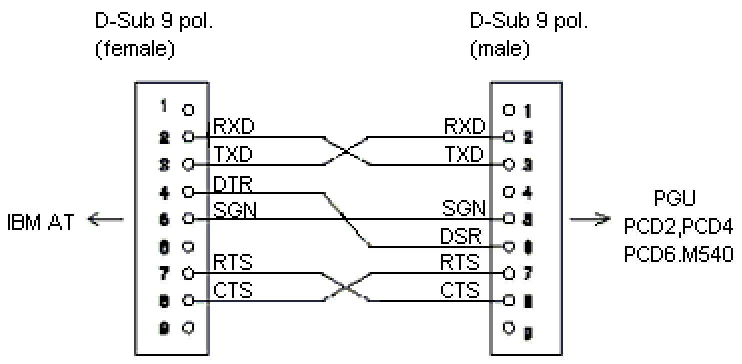


Figure 4. Cable Compatible with the PCD8.K111 Specification.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel using the S-BUS protocol is given below:

logical_channel_name=S-BUS*, id, port [, baud]*

where:	
S-BUS	- driver name;
id	- number of the controller in the S-BUS network;
port	- port name: COM1, COM2 etc.;
optional parameters:	
baud	- transmission channel.

If the optional parameters are not given, then by default it is assumed as follows:
transmission speed of 9600 baud.

EXAMPLE

The declaration of the logical channel named CHAN1 operating according to the S-BUS protocol and exchanging data with the controller numbered 1 through the COM2 port with a speed of 9600 baud is as follows:

CHAN1 = S-BUS, 1, COM2

The S-BUS driver is loaded as a DLL automatically.

Addressing the Process Variables

The syntax of symbolic address which is used for variables belonging to the S-BUS driver channel is as follows:

<type><index>

where:	
type	- variable type,
index	- index within the type.

Symbols of variable types (the type of raw variable value is given in parentheses):

C	- values of counters (DWORD),
F	- states of flags (WORD),
I	- values of inputs (WORD),
K	- current date and time in the form of an 8-bit array (BYTE),
O	- values of outputs (WORD),
RI	- values of registers treated as 32-bit fixed-point signed numbers (LONG),
RF	- values of registers treated as 32-bit floating-point numbers in SAIA format (FLOAT),
S	- status (WORD),
T	- values of timers (DWORD).

Values of variables of **C**, **F**, **O**, **RI**, **RF**, **T** that may be read and written.
Values of variables of **I**, **S** type that may be only read.
Values of variables of **K** types are used by the driver for time synchronization with a PCD.

The structure of **K** type variable buffer is as follows:

byte 0	- number of a week in a year,
byte 1	- number of a week day (Monday - 1, Sunday - 7),
byte 2	- two least significant digits of a year,
byte 3	- month,
byte 4	- day,
byte 5	- hour,
byte 6	- minute,
byte 7	- second.

Range of indexes for the **S** type is limited to 20 – 27.

<u>Variable defining the state of connection with the controller</u>

The variable is defined by address ON and takes value 1 when the last transmission is ended properly as well as when the last transmission is ended with failure. The variable is of WORD type and requires the NOTHING conversion function to be used.

EXAMPLE

Examples of variable declaration:

```
# values of registers treated as FLOAT
JJ_10,  RF1,  CHAN1, 1, 1,  NOTHING_FP

# values of registers treated as LONG
JJ_11,  RI11, CHAN1, 1, 1,  NOTHING_LONG

# states of flags
JJ_14,  F14,  CHAN1, 1, 1,  NOTHING

# values of inputs
JJ_14,  I14,  CHAN1, 1, 1,  NOTHING

# values of outputs
JJ_14,  O14,  CHAN1, 1, 1,  NOTHING
```

```
# values of counters
JJ_21,  C21,  CHAN1, 1, 1,  NOTHING_DW

# value of status
JJ_40,  S20,  CHAN1, 1, 1,  NOTHING
```

Driver Configuration

The S-BUS protocol driver may be configured by use of the **[S-BUS]** section placed in the application INI file. Individual parameters are transferred in separate items of the sections. Each item has the following syntax:

item_name=[number [,number]] [YES] [NO]

☑ ***ALARM=id,number***

Meaning	- driver of the S-BUS protocol may generate an alarm in case of loss and re-establishing the connection with the station. It is necessary in this case to create the item ALARM in the INI file.
Default value	- by default, the alarm is not generated.
Parameters:	
id	- number of the controller in the S-BUS network,
number	- number of the alarm to be generated in case of loss and re-establishing the connection.

☑ ***LOG_FILE=file_name***

Meaning	- the item allows to define a file to which all diagnostic messages of the S-BUS driver and the information about contents of telegrams received by the driver are written. If the item does not define the full path, then the log file is created in the current directory. The log file should be used only while the **asix** start-up.
Default value	- by default, the log file is not created.

☑ ***LOG_OF_TELEGRAMS=YES|NO***

Meaning	- the item allows to write to the log file (declared by use of the item LOG_FILE) the contents of telegrams sent within the communication with the controller. Writing the contents of telegrams to the log file should be used only while the **asix** start-up.
Default value	- by default, the telegrams are not written.

☑ ***LOG_FILE_SIZE =number***

Meaning	- the item allows to specify the log file size in MB.
Default value	- by default, the item assumes that the log file has a size of 1 MB.

☑ ***RECV_TIMEOUT=id,number***

Meaning	- the item allows to specify a maximal waiting time for arriving the first character of an answer from a given

	controller. After this time it is assumed that a given controller is switched off and the transmission session ends with an error.
Default value	- by default, it is assumed that the maximal waiting time for the first character of an answer is equal to 1000 milliseconds.
Parameters:	
id	- number of the controller in the S-BUS network,
number	- time in milliseconds (from 100 to 5000).

☑ ***CHAR_TIMEOUT=id,number***

Meaning	- the item allows to specify a maximal time between successive characters of an answer from a given controller. After this time it is assumed that the controller does not work correctly and the transmission session ends by an error.
Default value	- by default, it is assumed that the maximal time between successive characters of an answer is equal to 50 milliseconds.
Parameters:	
id	- number of the controller in the S-BUS network,
number	- time in milliseconds (from 10 to 300).

☑ ***ADDRESS_TIMEOUT=number***

Meaning	- the item allows to determine a time period between the character of address and the first character of data in an order sent to the PCD. The time period is necessary to switch over the PGU interface from the mode of address receiving to the mode of data receiving.
Default value	- by default, it is assumed that the time period between the address character and the first character of data is equal to 25 milliseconds.
Parameters:	
number	- time in milliseconds.

☑ ***NUMBER_OF_REPETITIONS=number***

Meaning	- the item allows to specify a number of repetitions in case of a transmission error.
Default value	- by default, the item assumes a values of 0 (no repetitions).

Time Synchronization Between the asix System and SAIA Controllers

In the S-BUS driver there is a mechanism of time synchronization between the **asix** system and SAIA controllers. The time synchronization is activated for each channel separately by means of items placed in the ASMEN section.

☑ ***TIME_SYNCHRONIZATION = channel, variable***

Parameters:	
channel	- name of a transmission channel used for communication with a given SAIA controller;
variable	- name of an ASMEN variable belonging to the channel CHANNEL and used for time synchronization.

The time synchronization consists on cyclic writing to the controller a frame containing an actual **asix** time. The frame is written by means of a built-in function for writing the S_BUS protocol time according to a frequency assigned to *variable*. The variable type must be the **K** type (clock support), the number of elements assigned to *variable* must accommodate the time frame, i.e. it must have a size of min. 8 bytes. As a conversion function the NOTHING_BYTE function must be used.

EXAMPLE

The definition of every-minute time synchronization for the channel CHAN1 by means of the variable SYNCHRO1:

```
[ASMEN]
DATA= SBUS.DAT
CHAN1 = S-BUS,0,COM1,9600
TIME_SYNCHRONIZATION = CHAN1, SYNCHRO1
```

The declaration of the variable SYNCHRO1 is found in the file *SBUS.DAT* and has the following form:

```
SYNCHRO1, clock synchronization,  K,   CHAN1,  8,  60,   NOTHING_BYTE
```

☑ ***MODE=id,mode_name***

Meaning	- up to now, the S-BUS protocol driver has serviced the PARITY transmission mode; from the version 1.02.000 the driver services also BREAK and DATA modes; settings of a proper mode are realized by the *MODE* parameter.
Default value	- omission of the item causes the PARITY mode realization.
Parameters:	
id	- number of controllers in the S-BUS network,
mode_name	- one of the following words: PARITY, BREAK or DATA.

NOTICE *Using a specific mode, one should remember about proper controller parametrization - the controller should use the same mode.*

Connection by Means of Modem

Driver of the S-BUS protocol is also able to exchange the data by means of a modem, also with use of the PGU interface.

<u>Connection by means of a modem is possible only by using the DATA transmission mode.</u>

S-BUS driver channel is the client of the AsComm server named S:BUS:*n*

where:

n	- it is the number of the serial port received from the ASMEN channel definition, e.g. if channel_name=S=BUS,1,com2,... then a client name is S-BUS:2.

To establish a connection on dial-up links by means of the AsComm program, the record given below must be placed in the [MODBUS:n] section:

Switched_line = Yes

If the modem is connected to the other port than COMn, then you should give the number of this port by means of the parameter *Port* or specify the modem name by means of the parameter *Modem*. You should also give a telephone number and define other required parameters. If MODBUS driver has to communicate with many controllers by means of the same modem, then one should define suitable number of channels taking the parameter *port* as a virtual transmission channel and place suitable number of sections in the INI file, by specifying in them an appropriate telephone number.

EXAMPLE

An example of the initialization file content:

```
[ASMEN]
....
Chan1 = MODBUS,1,COM11,9600,8,none,1,16,16
Chan2 = MODBUS,1,COM12,9600,8,none,1,16,16

[MODBUS:11]
Switched_line = Yes
Modem = US Robotics
Number = 11111111

[MODBUS:12]
Switched_line = Yes
Modem = US Robotics
Number = 22222222
```

In the example above Chan1 will communicate with a controller placed under the telephone number 11111111, and the Chan2 with a controller placed under the telephone number 22222222. The US Robotics modem will be used. The *Modem* parameter may be replaced by the parameter *Port*, which specifies the number of the serial port to which the modem is connected.

You should notice that the description of application of the MODBUS driver on switched links does not include any modem configuration guidelines. The modem configuration depends on modem types.

1.54. CtSbusTcpip - Driver of S-Bus Ethernet Protocol

Driver Use

The CtSbusTcpip driver is used for data exchange between **asix** system computers and PLCs of PCD SAIA-Burgess family by means of the Ethernet S-Bus protocol.

Declaration of Transmission Channel

The declaration of transmission channel using the CtSbusTcpip driver is as follows:

Channel= UNIDRIVER, CtSbusTcpip, *SbusNr=number; Port=number Server = IPaddress*
*[;*TimeSynchr = *number] [;*Timeout=*number]*

where:

UNIDRIVER	- universal **asix** system driver;
CtSbusTcpip	- name of the driver for communication with the PLCs of PCD SAIA-Burgess family;
SbusNr	- number of the controller in the S-BUS network;
Port	- number of the TCPIP port of the controller (by default 5050),
Server	- IP address of the controller,
TimeSynchr	- period (in seconds) for time synchronization with the controller – optional;
Timeout	- timeout (in milliseconds) between sending query and receiving response - optional.

EXAMPLE

An exemplary declaration of the channel for communication with the controller:
a. number in the S-BUS network - 3;
b. number of a TCPIP port - 5050;
c. IP address - 10.10.10.225;
d. time synchronization - every 20 seconds;

CHANNEL = UNIDRIVER, CtSbusTcpip, SbusNr=3; Port=5050;
Server=10.10.10.225; TimeSynchr =20

Declaration of Variables

The declaration of variables is the same as in the S-BUS driver. The syntax of the variable address is as follows:

<type><index>

where:

type	- variable type,
index	- indexes within the framework of the type.

The notations of variable types (the raw variable type is put in parentheses):

C	- counter values (DWORD),
F	- flag states (WORD),
I	- input states (WORD),
K	- current date & time in the form of 8-byte table (BYTE),
O	- output states (WORD),
RI	- values of registers treated as a 32-bit signed number (LONG),
RF	- values of registers treated as a 32-bit floating-point number in SAIA format (FLOAT),
S	- statuses (WORD),
T	- timer values (DWORD).

The variable values of the **C**, **F**, **O**, **RI**, **RF**, **T** type may be read and written.
The variable values of the **I**, **S** type may be only read.
The range of the indexes for the **S** type is from 20 to 27.

EXAMPLE

Examples of variable declarations.

```
# values of registers treated as FLOAT
JJ_10, , RF1, CHANNEL1, 1, 1, NOTHING_FP
# values of registers treated as LONG
JJ_11, , RI11, CHANNEL1, 1, 1, NOTHING_LONG
# flag states
JJ_14, , F14, CHANNEL1, 1, 1, NOTHING
# input states
JJ_14, , I14, CHANNEL1, 1, 1, NOTHING
# output states
JJ_14, , O14, CHANNEL1, 1, 1, NOTHING
# counter values
JJ_21, , C21, CHANNEL1, 1, 1, NOTHING_DW
# statuses values
JJ_40, , S20, CHANNEL1, 1, 1, NOTHING
```

Driver Configuration

The driver configuration is performed by using the separate section named **[CTSBUSTCPIP]**. By means of this section it is possible to declare:

- log file and its size,
- log of telegrams,
- PCD status verification.

☑ ***LOG_FILE=file_name***

Meaning	- it is a text file to which messages about the driver operation state are written; is used for diagnostic purposes.
Default value	- by default, the log file is not created.

Defining	- manual.

☑ ***LOG_FILE_SIZE=number***

Meaning	- allows to define the size of the log file.
Default value	- by default, the item assumes that the log file has a size of 1 MB.
Parameter:	
number	- size of the log file in MB.
Defining	- manual.

☑ ***LOG_OF_TELEGRAMS =YES | NO***

Meaning	- the item allows writing to the log file (declared with use of the LOG_FILE item) the contents of telegrams transmitted during the data exchange between the **asix** system and the controllers.
Default value	- NO.
Defining	- manual.

☑ ***WITHOUT_PCD_STATUS =YES | NO***

Meaning	- the item allows to control the variable status modification depending on the current controller status (PCD own status). If the item has the value YES, then the variable status is not dependent on the current variable of the controller status. If the item is set at value NO, then the variable status is dependent on the controller status - if it differs from 0x52 (RUN state), then the variable status is set at OPC_QUALITY_ COMM_FAILURE.
Default value	- NO.
Defining	- manual.

EXAMPLE

An exemplary driver section:

```
[CTSBUSTCPIP]
LOG_FILE=d:\tmp\ctsbustcpip\sbus.log
LOG_FILE_SIZE =20
LOG_OF_TELEGRAMS=YES
```

1.55. SINECH1 – Driver of Ethernet Network Protocol for SIMATIC S5 PLCs

Driver Use

The SINECH1 driver is used for data exchange between **asix** computers and SIMATIC S5 PLCs provided with the CP 1430, via an Ethernet network. An **asix** system computer must be provided with the SIEMENS CP1413 card.

The cooperation of the **asix** system with the PLC by use of the SINECH1 protocol needs developing the controller's program supporting the CP 1430.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel operating according to the SINECH1 protocol is given below:

logical_name=SINECH1*, fTsapPC, fTsapPLC, rTsapPC, rTsapPLC, MAC*

where:

fTsapPC	- name of TSAP used for the reading operation (FETCH) in the **asix** computer;
fTsapPLC	- name of TSAP used for the reading operation (FETCH) in the controller;
rTsapPC	- name of TSAP used for the writing operation (RECEIVE) in the **asix** computer;
rTsapPLC	- name of TSAP used for the writing operation (RECEIVE) in the controller;
MAC	- MAC network address assigned to the controller.

NOTE *Names of TSAPs are any 8-character strings. Names of TSAPs declared in the **asix** system must have their equivalents among TSAPs declared by means of COM 1430 on the controller side. Reading and writing from/to the controller is executed within separate connections.*

EXAMPLE

An exemplary declaration of transmission channel using the SINECH1 protocol is given below:

CHAN1=SINECH1,LocFetch,TestFetc,LocRecev,TestRece,08:00:06:01:00:2
2

The logical channel named CHAN1 has the following parameters defined:

SINECH1 protocol,	
LocFetch	- name of TSAP used for the reading operation (FETCH) in the **asix** system,
TestFetc	- name of TSAP used for the reading operation (FETCH) in the controller,
LocRecev	- name of TSAP used for the writing operation (RECEIVE) in the **asix** computer,
TestRece	- name of TSAP used for the writing operation (RECEIVE) in the controller,
08:00:06:01:00:22	- MAC network address assigned to the controller

Addressing the Process Variables

The rules of creating the symbolic addresses of variables belonging to the SINECH1 driver channel are the same as in case of the channel using the AS512 protocol

The SINECH1 driver is loaded as a DLL automatically.

Driver Configuration

Configuring the driver in the **[SINECH1]** section in the application INI file you use the following items.

☑ ***LOG_FILE=file_name***

Meaning	- the item allows to define a file to which all diagnostic messages of the SINECH1 driver and the information about contents of telegrams received by the driver are written. If the item does not define the full path, then the log file is created in the current directory. The log file should be used only while the **asix** start-up.
Default value	- by default, the log file is not created.

☑ ***LOG_OF_TELEGRAMS=YES|NO***

Meaning	- the item allows to write to the log file (declared by use of the item LOG_FILE) the contents of telegrams sent within the communication with the controller. Writing the contents of telegrams to the log file should be used only while the **asix** start-up.
Default value	- by default, the telegrams are not written.

☑ ***LOG_FILE_SIZE =number***

Meaning	- the item allows to specify the log file size in MB.
Default value	- by default, the item assumes that the log file has a size of 1 MB.

1.56. SINECL2 – Driver of PROFIBUS Protocol for SIMATIC S5 PLCs

Driver Use

The SINECL2 driver is used for data exchange between **asix** computers and SIEMENS SIMATIC S5 PLCs provided with the CP5430 card, with the aid of the SINEC L2 local network based on the FDL protocol (2-nd level of the PROFIBUS standard). An **asix** system computer must be provided with the CP5412 (A2) or CP5613 communication processor and system software used to support this processor.

The software of the controller dedicated for operation together with the **asix** system must meet the following requirements:

- program in the controller must contain calls of function blocks executing receiving and sending telegrams through the CP5430 according to the FDL protocol;
- number of the node given to the CP5430 must be unique within the local SINEC L2 network;
- parameters of the CP5430 and CP5412(A2) or CP5613 operation must be compatible.

In each transmission channel established between any **asix** system computer and any CP5430 communication processor, it is allowed to use 10 subchannels, multiplying in this way a quantity of transferred information.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel operating according to the SINEC L2 protocol is given below:

logical_name=SINECL2,*PC_node*,*PLC_node [,subchannel]*

where:

PC_node	- number of the node assigned to the application computer;
PLC_node	- number of the node assigned to the CP5430 processor in the controller with which the connection is to be executed in a given transmission channel;

subchannel	- numbers of subchannels which are used in a given transmission channel. An absence of this parameter signifies that all the subchannels (from 1 to 10 inclusive) are used in a given channel.

EXAMPLE

Exemplary items declaring transmission channels working according to the SINEC L2 protocol are given below:

```
CHAN2=SINECL2,5,1,1,2,3,4
CHAN3=SINECL2,5,2
```

The transmission channel with the logical name CHAN2 has the following parameters defined:

- SINECL2 protocol using the SINEC L2 local network;
- number of the node assigned to the computer - 5;
- number of the node assigned to the CP5430 communication processor - 1;
- used subchannels with numbers 1, 2, 3 and 4.

The transmission channel with the logical name CHAN3 has the following parameters defined:

- SINECL2 protocol using the SINEC L2 local network;
- number of the node assigned to the computer - 5;
- number of the node assigned to the CP5430 communication processor - 2;
- all the subchannels are used.

Addressing the Process Variables

The rules of creating the symbolic addresses belonging to the SINECL2 type channel are the same ones as for AS512 type channels.

The SINECL2 protocol driver requires installing the CP5412(A2) or CP5613 card and the driver of this card supplied in the DP-5412/Windows NT packet by SIEMENS.

The SINECL2 protocol driver is loaded as a DLL automatically.

Driver Configuration

☑ ***ALARM_LOG = [YES|NO]***

Meaning	- declaration of writing to the log file the messages about an arrival of alarm telegrams.
Default value	- NO.
Defining	- manual.

☑ ***ERROR_LOG = [YES|NO]***

Meaning	- declaration of writing to the log file the messages about transmission errors.
Default value	- NO.
Defining	- manual.

☑ ***MAX_NOUMBER_OF_NODES = number***

Meaning	- declaration of maximal number of nodes in the SINECL2 network.
Default value	- 16.
Defining	- manual.

☑ ***SAP = id, PC_SAP, PLC_SAP***

Meaning	- declaration of mapping the SAPs, creating a logical channel named *id,* on the PC and PLC sides;
Default value	- by default, 10 pairs of SAPs mapped as follows (id, PC_SAP, PLC_SAP) are used:

```
1,  35,  45
2,  36,  46
3,  37,  47
4,  38,  48
5,  39,  49
6,  40,  50
7,  41,  51
8,  42,  52
9,  43,  53
10, 44,  54
```

Defining	- manual.

☑ ***STATISTICS =[YES|NO]***

Meaning	- transmission statistics writing to the log file every minute.
Default value	- NO.
Defining	- manual.

☑ ***TIMEOUT =number***

Meaning	- timeout for an answer from the controller expressed in ticks of duration time equal to 400 ms.
Default value	- 3.
Defining	- manual.

☑ ***LOG_FILE = name***

Meaning	- declaration of the log file name with diagnostic messages of the SINECL2 driver.
Default value	- log file is not created.
Defining	- manual.

☑ ***CP_TYPE = number***

Meaning	- declaration of the communication processor card type used in the **asix** system computer. Two types are allowable: 1/ CP5613 identified by the number 5613; 2/ CP4512 (A2) identified by the number 5412.
Default value	- 5412 (CP5412 (A2) communication processor).
Defining	- manual.

☑ ***DELAY = number***

Meaning	- timeout declaration (in milliseconds) after an occurrence of SDA telegram sending error.
Default value	- 0.
Defining	- manual.

1.57. CtSi400 - Driver of Protocol for Sintony Si 400 Alarm Central of SIEMENS

Driver Use

The CtSi400 driver is used for data exchange between **asix** computers and a Sintony Si 400 alarm central of SIEMENS. The communication is executed by means of serial links in the RS-232 standard.

Additional recommendations:
For proper operation of the alarm system it is demanded to connect only one visualization computer to a Sintony central (despite the central is equipped with 3 interfaces, during the system operation it is possible to use only one of them). The best solution is to connect a computer to the J7 interface with use of a SAQ11 cable recommended by SIEMENS.

Declaration of Transmission Channel

The CtSi400 driver is loaded by the universal **asix** system driver - UNIDRIVER.

The syntax to declare the transmission channel operating with the CtSi400 driver is given below:

Channel=UNIDRIVER, CtSi400, *Port=port_nr* [;*BaudRate=baud*] [;*Timeout=ms_number*] [;*AlarmOffset=offset*]*USERID=user_id*

where:

UNIDRIVER	- the universal **asix** system driver;
CtSi400	- the driver used for communication with a SINTONY SI 400 central;
port_nr	- serial port number (for COM1 it passed 1, for COM2 is passed 2, etc.);
baud	- transmission speed passed in bauds. By default, it is assumed 9600 Bd;
ms_number	- timeout of waiting for the controller response (in milliseconds); it is passed 1000 milliseconds by default;
offset	- offset added to the number of each alarm sent from a SINTONY SI 400 central;
user_id	- user identifier sent to SINTONY SI 400 in control frames. Proper values include in the range 0-500.

By default, it is assumed:
- number of character bits - 8,
- without parity check (NONE),

- number of stop bits - 1.

EXAMPLE

An exemplary declaration of the channel using CtSi400 on the serial port COM2 with the receiving timeout that equals 2000 ms and the offset, added to each alarm number, equaling 1500 is as follows:

PLC1 = UNIDRIVER, CTSI400, Port=2; Timeout=2000; OffsetAlarm=1500

Addressing Process Variables

The syntax of the symbolic address for all variables used to monitor the central state is given below:

<type><index>

and for variables used to control:

<type>

or

<type><index>

or

<type><p_index>.<r_index>

where:

type	- variable type,
index	- index within the type,
p_index	- partition number (0-15),
r_index	- room's number within the partition (0-7).

Types of variables used to monitor

The are several types of variables used to monitor, which are listed below. The driver permits readout operations to be performed on these variables, while all write operations end with the OPC_E_BADRIGHTS error.

The bit 0 means the least significant bit in the following list.

- **IS<*index*>** - input status

PState	- bits 7-6
Lstate	- bits 5-3
Reserved	- bits 2-0

- **IP<*index*>** - input parameters:

PSL	- bits 15-14
V.Address	- bits 13-8
Type	- bits 7-4,
SubType,	- bits 3-1

- **IA<*index*>** - input address

room nr - bits 12-9
partition nr - bits 4-0

- **IN<*index*>** - input name

- **OS<*index*>** - output status

State - bit 7
OutInTest - bit6
BlinkBit - bit 5
NbrPulse - bit 4
Reserved - bits 3-0

- **OP<*index*>** - output parameters

Ltype - bits 7-0

- **OA<*index*>** - output address

room nr - bits 12-9
partition nr - bits 4-0

- **ON<*index*>** - output name

- **RP<*index*>** - room parameters

Rset - bits 15-8 (room1 is in the bit no. 15, room2 is in the bit no. 14, etc.)
Rarm - bits 7-0 (room1 is in the bit no. 7, room2 is in the bit no. 6, etc.)

- **RS<*index*>** - room status

room1 - bits 15-14
room2 - bits 13-12

etc.

- **PS<*index*>** - partition status

TAMP - bit 15
PA - bit 14
Fire - bit 13
AM - bit 12
Chime - bit 11
Mess - bit 10
Reserved - bits 9-8
Pstate - bits 7-6
PFA - bit 5
PPA - bit 4
PFS - bit 3
PPS - bit 2
PUS - bit 1
BA - bit 0

- **SS<*index*>** - system status

ParamCh	- bit 15
Evt Overflow	- bit 14
Avzone	- bits 13 - 8
Mains	- bit 7
Batt	- bit 6
Fuse	- bit 5
Line	- bit 4
CMS1	- bit 3
CMS2	- bit 2
InComm	- bit 1
Evt queue	- bit 0

Types of variables used to control

The are several types of variables used to control, which are listed below. The driver permits write operations to be performed on these variables, while all readout operations end with the OPC_E_BADRIGHTS error.

- **cIB<*index*>** - input bypass (index = 0..0xFFFF)

Writing any value to the variable will cause the performance of the command for the input with the *index* number.

- **cIS<*index*>** - input in soak test (index = 0..0xFFFF)

Writing any value to the variable will cause the performance of the command for the input with the *index* number.

- **cOT<*index*>** - set output in test mode (index = 0..0xFFFF)

Writing any value to the variable will cause the performance of the command for the output with the *index* number.

- **cTO<*index*>** - toggle output (index = 0..0xFFFF)

Writing any value to the variable will cause the performance of the command for the output with the *index* number.

- **cPP<*p_index*>** - partition part-set (p_index = 0..0xF)

Writing any value to the variable will cause the performance of the command for the partition with the *p_index* number.

- **cPF<*p_index*>** - partition full-set (p_index = 0..0xF)

Writing any value to the variable will cause the performance of the command for the partition with the *p_index* number.

- **cPU<*p_index*>** - partition unset (p_index = 0..0xF)

Writing any value to the variable will cause the performance of the command for the partition with the *p_index* number.

- **cCA<*p_index*>** - clear alarm memory in partition (p_index = 0..0xF)

Writing any value to the variable will cause the performance of the command for the partition with the *p_index* number.

- **cRF<*p_index*>.<*r_index*>** - room full-set (p_index = 0..0xF, r_index = 0..0xF)

Writing any value to the variable will cause the performance of the command for the room with the *r_index* number from the partition with the *p_index* number.

- **cRU<*p_index*>.<*r_index*>** - room unset (p_index = 0..0xF, r_index = 0..0xF)

Writing any value to the variable will cause the performance of the command for the room with the *r_index* number from the partition with the *p_index* number.

- **cSR<*p_index*>** - partition reset sounders/bells (p_index = 0..0xF)

Writing any value to the variable will cause the performance of the command for the partition with the *p_index* number.

- **cIP<*index*>** - input activation pulse (index = 0..0xFFFF)

Writing any value to the variable will cause the performance of the command for the input with the *index* number.

- **cIL<*index*>** - input activation latch (index = 0..0xFFFF)

A written value must convert to a double type. If the value 0 is received after conversion, for the input with the *index* number there will be performed the command with the parameter *commutation level* equaling *0*; otherwise the parameter will be equal to 1.

- **cAV** - new current A/Video zone

A written value must be an integer from the range <0; 32>. Writing the value 0 will activate the *autoswitch* option.

- **cTC** - transmission to CMS

A written value must have the value 1 or 2 and indicate CMS1 or CMS2 suitably.

- **cER** - reset the pending event buffer

Writing any value to the variable will cause the performance of the command.

IN and ON variables are 16-element BYTE type arrays. Other variables are WORD type values.

All the variables used to control are of the WORD type.

EXAMPLE

Exemplary variable declarations:

```
JJ_00, input status 1,        IS1,       PLC1, 1, 1, NIC
JJ_01, output status 10,      OS10,      PLC1, 1, 1, NIC
JJ_02, input 2 name,          IN2,       PLC1, 16, 1, NIC_BYTE
JJ_03, system status,         SS1,       PLC1, 1, 1, NIC
```

cIB3, input 3 bypass,	cIB3,	PLC1, 1, 1, NIC
cRU1.2, part 1 room 2 unset,	cRU1.2,	PLC1, 1, 1, NIC
cAV, current A/V zone,	cAV,	PLC1, 1, 1, NIC

Time Stamp

All values of variables read from a SINTONY SI 400 central are stamped with the local PC time.

Driver Configuration

The driver configuration is performed by using the separate section named [**CTSI400**]. By means of this section it is possible to declare:
- log file and its size,
- log of telegrams,
- delay between opening a session and data exchange.

☑ ***LOG_FILE = file_name***

Meaning	- the item allows to define a file to which all diagnostic messages of the driver will be written. If the item does not define the full path, then the log file is created in the current directory. The log file should be used only while the **asix** system start-up.
Default value	- by default, the log file is not created.
Defining	- manual.

☑ ***LOG_FILE_SIZE = number***

Meaning	- allows to specify the log file size.
Default value	- 1MB.
Defining	- manual.
Paraneter:	
number	- number in MB.

☑ ***LOG_OF_TELEGRAMS=[YES|NO]***

Meaning	- item allows to write to the log file (declared bye use of the item LOG_FILE) the contents of telegrams received by the driver. Writing the contents of telegrams to the log file should be used only in the stage of the **asix** system start-up.
Default value	- NO.
Defining	- manual.

☑ ***DELAY_AFTER_OPEN_SESSION=number***

Meaning	- allows to define a timeout between opening a session and sending the first query about data within this session.
Default value	- by default, the value is set to 100.
Defining	- manual.
Parameter:	

Number	- timeout value in milliseconds.

1.58. CtSNPX - Driver of SNPX Protocol for GE Fanuc PLCs

Driver Use

The driver of the SNPX protocol (Series Ninety Protocol) is used for data exchange between **asix** system computers and GE_FANUC 90-30 PLCs as well as GE_FANUC 90 CMM and PCM modules. The communication is executed by means of serial links.

The driver allows access to the following controller variable types:

- Discrete Inputs (%I),
- Discrete Outputs (%O),
- Discrete Internals (%M),
- Analog Inputs (%AI),
- Analog Outputs (%AO),
- Registers (%R),
- Genius Global Data (%G).

The driver does not handle the following controller variable types (system and temporary types):

- %SA Discrete,
- %SB Discrete,
- %SC Discrete,
- %S Discrete,
- Discrete Temporary (%T).

Declaration of Transmission Channel

The syntax of declaration of transmission channel operating according to the SNPX protocol is as follows:

Channel=UNIDRIVER, CtSNPX, *Port=number; [Baudrate=number;]*
[ParityBit=check_parity_name;]
[TimeSynchrX=address[:period];] [T4=timeout_break;]
[T2=response_timeout;] [TBroadCast=timeout_broadcast]

where:

UNIDRIVER	- universal **asix** system driver;
CtSNPX	- driver name;
Port	- number of the COM serial port;
BaudRate	- transmission speed between the computer and the controller; there are the following acceptable values: 300, 600,1200,2400, 4800, 9600, 19200 Bd; a default value – 19200 Bd;

ParityBit	- determines the method of frame parity check; there are the following acceptable values: *NONE*, *ODD*, *EVEN*; default value – ODD (odd parity check);
T4	- timeout (in milliseconds) between sending BREAK and BROADCAST ATTACH; a default value – 50 millisecodns;
TBroadCast	- timeout between sending BROADCAST ATTACH and sending the first request to the controller; a default value – 2000 milliseconds;
T2	- timeout (in milliseconds) for receiving the first bit of the response; a default value – 2000 milliseconds;
SynchrCzasuX	- cyclic (every *period* in seconds) date & time frame write at the given address in the controller; there are 99 positions of time synchronization from the range of names from *SynchrCzasu1*to *SynchrCzasu99*; if the parameter period is not given, the synchronization is performed every 60 seconds by default; the date & time frame synchronization is compatible with *SVCREQ 7* – the procedure of date & time write:

```
struct   dateTime
 {
  byte   Year;
  byte   Month;
  byte   Day;
  byte   Hour;
  byte   Minute;
  byte   Second;
  byte   DayOfWeek;
  byte   NotUsed;       // always 0
  word   wSynchr;       // set to 1 at new date & time
frame write
 };
```

NOTICE *The parameters given in the channel declaration must be compatible with the parameters set for communication ports of the controllers handled by this channel.*

EXAMPLE

An exemplary declaration of the channel, in which the controllers with identifiers A123 and B456 are handled, is given below:
1/ for the controller with the A123 identifier - time is synchronized by writing to the register area beginning with R10 (every 25 seconds),
2/ for the controller with the B456 identifier - time is synchronized by writing to the register area beginning with R10 (with default frequency).

The communication with the controllers is performed over COM2 by means of standard transmission parameters, i.e. 19200 Bd, odd parity control, the first bit of stop and standard timeouts of the SNPX protocol.

CHANNEL = UNIDRIVER, CtSNPX, Port=2; TimeSynchr1=A123.R10:25; TimeSynchr2=B456.R20

Declaration of Variables

The driver makes the following variable types available:

I	- Discrete Input (%I) in BIT mode,

IB	- Discrete Input (%I) in BYTE mode,
IW	- Discrete Input (%I) in WORD mode,
Q	- Discrete Output (%I) in BIT mode,
QB	- Discrete Output (%I) in BYTE mode,
QW	- Discrete Output (%I) in WORD mode,
M	- Discrete Internal (%I) in BIT mode,
MB	- Discrete Internal (%I) in BYTE mode,
MW	- Discrete Internal (%I) in WORD mode,
G	- Genius Global Data (%G) in BIT mode,
GB	- Genius Global Data (%G) in BYTE mode,
GW	- Genius Global Data (%G) in WORD mode,
AI	- Analog Input (%AI) in WORD mode,
AO	- Analog Output (%AO) in WORD mode,
R	- Register (%R) treated as WORD,
RL	- two following Registers (%R) treated as DWORD,
RF	- two following Registers (%R) treated as FLOAT,

The syntax of the variable address is as follows:

[<*CpuID*>.]<*Type*><*Index*>

where:

CpuID	- CPU identifier;
Type	- variable type name;
Index	- variable address within the framework of the variable type *Type*.

NOTICE
1. *CpuID* may be omitted in the variable address exclusively when only one controller is connected to the serial link. In such case, commands sent to the controller contain the identifier set at NULL (the real identifier set in the controller is unimportant).
2. *Index* indicates bit number, from which the range of bits ascribed to the variable begins, for discrete variables. Index may have one of the following values (by pattern of addressing used in VersaPro), depending on mode of making discrete variables available:
a/ for BIT mode - any value w.e.f. 1,
b/ for BYTE mode - value 1, 9, 17, etc. (numbers of the first bit of successive bytes),
c/ for WORD mode - value 1, 17, 33, etc. (numbers of the first bit of successive words).

EXAMPLE

An exemplary variable declaration (the variable values come form the controllers identified by A123 and B456):

```
JJ_01, Register R3,                A123.R3,    CHANNEL, 1, 1, NOTHING
JJ_02, Analog Input 1,             A123.AI1,   CHANNEL, 1, 1, NOTHING
JJ_03, Discrete Input 3,           B456.I3,    CHANNEL, 1, 1, NOTHING
JJ_04, Discrete Output Byte 9 ,    A123.QB9,   CHANNEL,1,1, NOTHING_BYTE
JJ_05, Genius Global Word 17 ,     A123.GW17,  CHANNEL, 1, 1, NOTHING
JJ_06, Discrete Internal Word 33,  B456.MW33,  CHANNEL, 1, 1, NOTHING
```

Driver Configuration

The driver configuration is performed by using the separate section named **[CTSNPX**]. By means of this section it is possible to declare:

- log file and its size,
- log of telegrams.

☑ ***LOG_FILE=file_name***

Meaning	- the item allows to define a file to which all the diagnostic messages of the driver will be written.
Default value	- by default, the log file is not created.

☑ ***LOG_FILE_SIZE =number***

Meaning	- the item allows to define the size of the log file in MB.
Default value	- by default, the item assumes that the log file has a size of 1 MB.
Parameter:	
number	- size of the log file in MB.

☑ ***LOG_OF_TELEGRAMS =YES | NO***

Meaning	- the item allows writing to the log file (declared with use of the LOG_FILE item) the contents of telegrams transmitted during the data exchange between the **asix** system and controllers.
Default value	- NO.

EXAMPLE

An exemplary driver section:

```
[CTSNPX]
LOG_FILE=d:\tmp\ctLG\LG.log
LOG_FILE_SIZE=3
LOG_OF_TELEGRAMS=YES
```

1.59. SPA - Driver of SPA Protocol

Driver Use

The SPA driver is used for data exchange between devices manufactured by ABB connected to the SPA bus and an **asix** system computer. The communication is executed by means of serial interfaces in the RS232C or RS485 standard.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel operating according to the SPA protocol is given below:

logical_channel_name=SPA, *number*, *type*, *port*, *baud*, *AlTxtOff*, *AlValOff*, *password*

where:

SPA	- driver name;
number	- number assigned to a remote device;
type	- remote device type: 1 – SPAJ 141C, 2 – SPAM 150C,
port	- serial port name;
baud	- transmission speed: 9600 or 4800 – it must be compatible to settings in the remote device;
AlTxtOff	- number added to the number of the text event read from the remote device in order to build an unique alarm number transferred to the **asix** system;
AlValOff	- number added to the event number in order to build an unique alarm number transferred to the **asix** system;
password	- password allowing write operations to the remote device – it must be compatible to settings in the remote device.

EXAMPLE

The declaration of the logical channel named CHAN1 operating according to the SPA protocol and with parameters as below:

- remote device number - 4
- device type - SPAM 150 C
- port - COM1
- transmission speed - 9600 Bd
- number added to the number of a text event - 100
- number added to the number of an event with a value - 200
- password - 123

is as follows:

```
CHAN1=SPA, 4, 2, COM1, 9600, 100, 200, 123
```

The SPA driver is loaded as a DLL automatically.

Addressing the Process Variables

The syntax of symbolic address which is used for variables belonging to the SPA driver channel is as follows:

<variable_type><channel>.<index>

where:

variable_type	- type of the process variable;
channel	- channel number in the device from which the process variable is taken;
index	- process variable index within the type.

Types of process variables:

I	- values of I category data,
O	- values of O category data,
S	- values of S category data,
V	- values of V category data.

The range of used channels, types of supplied process variables, index range within each of types and meaning of individual elements within the type are specific for the remote device type.
A detail specification is included in the documentation of the remote device.

NOTE *Raw values of all process variables are of FLOAT type.*

EXAMPLE

An example of the declaration of variables for the SPAM 150 C device (according to the documentation all the variables are placed in the channel no. 0):

```
X1,  current in phase L1 ,                  I0.1,  CHAN1,  1,  1,  NOTHING_FP
X2,  stimulation of stage Io> ,             O0.8,CHAN1,  1,  1,  NOTHING_FP
X3,  coefficient p for thermal unit, S0.3,         CHAN1,  1,  1,  NOTHING_FP
X4,  current I measured during stimulation,V0.21,CHAN1,  1,  1,  NOTHING_FP
X5,  current I measured during activation,0.41,CHAN1,  1,  1,  NOTHING_FP
```

Generating the Alarms

Numbers of events, generated by remote devices, have the same variation range. In order to specify unambiguously which device the event under consideration come from, the SPA driver adds to the event number a number specified in the channel declaration as AllTxtOff (for text events) or AllValOff (for events with a value). In this way created number is transferred to the **asix** system as an alarm number.

Beside an alarm number, the SPA driver transfers a number of the remote device from which a given event comes. The device number may be used in a message related to the alarm by giving a formatting string (%3.0f) in the content of the alarm message.

For transferring alarms the SPA driver uses the function *AsixAddAlarmGlobalMili()* by default. The item GLOBAL_ALARMS allows to change the default settings and to transfer the alarms by means of the *function AsixAddAlarmMili()*.

Types of SPA remote devices implemented in the SPA driver generate only text events.

Driver Configuration

The SPA protocol driver may be configured by use of the **[SPA]** section placed in the application INI file. Individual parameters are transferred in separate items of the sections. Each item has the following syntax:

item_name=[number [,number]] [YES|NO]

☑ ***LOG_FILE=file_name***

Meaning	- the item allows to define a file to which all diagnostic messages of the SPA driver and information about contents of telegrams received and sent by the SPA driver will be written. If the item does not define the full path, then the log file is created in the current directory. The log file should be used only while the **asix** system start-up.
Default value	- by default, the log file is not created.
Defining	- manual.

☑ ***LOG_OF_TELEGRAMS=YES|NO***

Meaning	- the item allows to write to the log file (declared by use of the item LOG_FILE) the contents of telegrams sent and received from the SPA bus within reading the process variables. Writing the contents of telegrams to the log file should be used only while the **asix** system start-up.
Default value	- by default, telegrams are not written.
Defining	- manual.

☑ ***TRANSMISSION_DELAY=number***

Meaning	- the item allows to determine a time interval (as a multiple of 10 milliseconds) between two successive operations on the SPA bus.
Default value	- by default, the item assumes a value of 1 (10 milliseconds).
Defining	- manual.

☑ ***NUMBER_OF_REPETITIONS=number***

Meaning	- the item allows to specify a number of repetitions in case of a transmission error.
Default value	- by default, the item assumes a value of 0 (no repetitions).
Defining	- manual.

☑ ***DATA_UPDATE=number***

Meaning	- the item allows to specify a time period (in seconds) after exceeding of which you should update values of process variables stored in internal buffers of the driver.
Default value	- by default, the item assumes a value of 5.
Defining	- manual.

☑ ***TIME_UPDATE=number***

Meaning	- the item allows to specify a time period (in seconds) after exceeding of which you should send an actual time to remote devices.
Default value	- by default, the item assumes a value of 1.
Defining	- manual.

☑ ***DATE_UPDATE=number***

Meaning	- the item allows to specify a time period (in seconds) after exceeding of which you should send an actual date to remote devices.
Default value	- by default, the item assumes a value of 30.
Defining	- manual.

☑ ***ALARM_UPDATE=number***

Meaning	- the item allows to specify a time period (in seconds) which separates the successive cycles of reading alarm buffers of all remote devices supported by individual serial interfaces.
Default value	- by default, the item assumes a value of 1.
Defining	- manual.

☑ ***CHECKSUM=YES|NO***

Meaning	- the item allows to control the building of checksum in telegrams sent to the SPA bus. If the item has the value

	NO, then two characters X are inserted in telegram instead of the checksum.
Default value	- by default, the checksum is built.
Defining	- manual.

☑ ***TELEGRAM_EXCLUSION=YES|NO***

Meaning	- the item allows to exclude, from the list of supported telegrams, telegrams which are stamped by an addressed device with a code N (illegal range of variables in the telegram or not supported type of variables). The exclusion of telegrams allows to use interfaces efficiently.
Default value	- by default, telegrams are excluded.
Defining	- manual.

☑ ***GLOBAL_ALARMS=YES|NO***

Meaning	- the item controls the way of transferring alarms read from remote devices to the alarm system in the **asix** system.
Default value	- default alarms are transferred to the alarm system as global alarms (transferred to the alarm system by means of the function *AsixAddAlarmGlobalMili()*). Setting the value of the item GLOBAL_ALARMS on NO causes that alarms are transferred to the alarm system by means of the function *AsixAddAlarmMili()*.
Defining	- manual.

1.60. SRTP - Driver of SRTP Protocol

Driver Use

The SRTP driver is designed for data exchange between the **asix** system and the GE Fanuc Automation PLCs of VersaMax Nano/Micro, WersaMax and 90 series, by means of SRTP (Service Request Transfer Protocol) using an Ethernet network with the TCP/IP protocol.

The data exchange with the VersaMax Nano/Micro PLCs is realized with the aid of the IC200SET001 converter.

The communication with the WersaMax and 90 PLCs demands the IC693CMM321 converter.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel operating according to the SRTP protocol is given below:

channel_name=SRTP, *IP_address [, port]*

where:

SRTP	- driver name;
IP_driver	- IP address assigned to the IC693CMM321communication module;
port	- optional port number through which the connection with the IC693CMM321 communication module is executed (by default, 18245).

For each IC693CMM321 module a separate transmission channel declaration is required.

EXAMPLE

An exemplary declaration of the transmission channel CHANNEL used for communication with the IC693CMM321 module. The channel has the 10.10.10.70 IP address and uses a default port numbered 18245.

```
CHANNEL = SRTP, 10.10.10.70
```

Types of Process Variables

In the driver the following types of process variables are defined:

I	- Discrete Input (%I) in BIT mode,
IB	- Discrete Input (%I) in BYTE mode,
IW	- Discrete Input (%I) in WORD mode,
Q	- Discrete Output (%Q) in BIT mode,
QB	- Discrete Output (%Q) in BYTE mode,
QW	- Discrete Output (%Q) in WORD mode,
M	- Discrete Internal (%M) in BIT mode,
MB	- Discrete Internal (%M) in BYTE mode,
MW	- Discrete Internal (%M) in WORD mode,
G	- Genius Global Data (%G) in BIT mode,
GB	- Genius Global Data (%G) in BYTE mode,
GW	- Genius Global Data (%G) in WORD mode,
AI	- Analog Input (%AI) in WORD mode,
AQ	- Analog Output (%AQ) in WORD mode,
R	- Register (%R) treated as WORD,
RL	- two successive Registers (%R) treated as DWORD,
RF	- two successive Registers (%R) treated as FLOAT,
TD	- current time and date of the controller.

Values of **I**, **IB**, **IW** and **AI** type variables may be only read whereas values of the other variables may be read and written.

The following types of variables (system, temporary) are NOT supported:

- Discrete SA (%SA),
- Discrete SB (%SB),
- Discrete SC (%SC),
- Discrete S (%S),
- Discrete Temporary (%T).

Addressing the Process Variables

The syntax of symbolic address which is used for variables belonging to the SRTP driver channel is as follows:

<Type><Index>

where:

Type	- name of the variable type;
Index	- variable address within the type *Type* of the variable.

For discrete variables (**I**, **IB**, **IW**, **Q**, **QB**, **QW**, **M**, **MB**, **MW, G**, **GB**, **GW**) *Index* indicates the number of bit from which the range of bits assigned to the variable begins. Depending on the mode of accessing the discrete variables, *Index* may assumes the following values:

a/ for BIT mode	- any value beginning from 1;
b/ for BYTE mode	- values 1, 9, 17, etc. (numbers of the first bit of successive bytes);
c/ for WORD mode	- values 1, 17, 33, etc. (numbers of the first bit of successive words).

EXAMPLE

Examples of variable declarations:

```
JJ_1,  %R1 and %R2 as FLOAT,           RF1,     CHANNEL,1,1,NOTHING_FP
JJ_3,  %R3 and %R4 as DWORD,           RF3,     CHANNEL,1,1,NOTHING_DW
JJ_5,  %R5 as WORD,                    R5,      CHANNEL,1,1,NOTHING
JJ_31, single bit %M1,                 M1,      CHANNEL,1,1,NOTHING
JJ_32, bits %Q9 - %Q16 as one byte,    QB9,     CHANNEL,1,1,NOTHING_BYTE
JJ_33, bits %I17 - %I32 as one word,  IW17,     CHANNEL,1,1,NOTHING
JJ_40, writing date and time to PLC,   TD,      CHANNEL,8,15,NOTHING_BYTE
```

Date and Time Synchronization with the Controller

A mechanism of date and time synchronization between the **asix** system and GE Fanuc PLCs is built in the driver. The synchronization is performed for each transmission channel separately by means of items placed in the ASMEN section:

☑ ***TIME_SYNCHRONIZATION = channel, variable***

Parameters:

channel	- name of the transmission channel used for communication with a specified IC693CMM321 module;
variable	- name of the ASMEN variable, belonging to the channel CHANNEL, used for date and time synchronization.

The date and time synchronization consists in a cyclic writing to the controller an actual frame containing the current date and time of **asix**. The frame is written by use of a built-in SRTP protocol function for writing date and time according to the frequency assigned to *variable*. The variable type must be the **TD** type (support of date and time), number of elements assigned to *variable* must have a size of 8 (size of date and time frame). The function NOTHING_BYTE must be used as the conversion function.

An exemplary definition of every-minute time synchronization for the channel CHANNEL by use of the variable SYNCHRO is given below:

```
[ASMEN]
DATA= SRTP.DAT
CHANNEL = SRPT, 10.10.10.70
TIME_SYNCHRONIZATION = CHANNEL, SYNCHRO
```

The declaration of the variable SYNCHRO may be found in the file *SRTP.DAT* and is as follows:

```
SYNCHRO, date and time synchronization, TD,    CHANNEL,  8,  60,
NOTHING_BYTE
```

Driver Configuration

The driver configuration is executed by use the separate section named **[SRTP]**. By means of this section is possible to declare:

- waiting timeout for an answer from a slave type device,
- log file and its size,
- log of telegrams,
- time for establishing connections.

☑ ***LOG_FILE = file_name***

Meaning	- allows to define a file to which all diagnostic driver messages and information about the content of telegrams received by the driver will be written. If the item LOG_FILE does not define the full path, then the log file will be created in the current directory. The log file should be used only in the stage of the **asix** start-up.
Default value	- log file is not created.
Defining	- manual.

☑ ***LOG_FILE_SIZE = number***

Meaning	- declares a log file size in MB.
Default value	- 1MB.
Defining	- manual.

☑ ***RECV_TIMEOUT = client_IP_address, number***

Meaning	- for each IC693CMM321 module, the maximal time which may elapse between sending a query and receiving an answer (so called receiving timeout). After having exceeded the timeout the connection will be broken (and re-established). The timeout value is determined individually for each IC693CMM321 module.
Default value	- 5s.
Defining	- manual.

Parameters:

client_IP_address	- IP address of the IC693CMM321 module;
number	- timeout value in seconds.

☑ ***LOG_OF_TELEGRAMS = [YES/NO]***

Meaning	- allows to write, to the log file (declared by use of the item LOG_FILE), contents of telegrams sent/received by the driver. Telegrams content write should be used only in the stage of the **asix** system start-up.
Default value	- NO.
Defining	- manual.

☑ ***STARTUP_TIME = number***

Meaning	- allows declaring a time (in seconds) provided for establishing the network connections with all the IC693CMM321 modules on the stage of the asix start-up.
Default value	- 3s.
Defining	- manual.

1.61. TALAS - Driver of TALAS Analyzer Protocol

Driver Use

The TALAS driver is used for data exchange between TALAS emission computers and the **asix** system by use of serial interfaces. The driver was created in order to operate with devices which have a firm software v 2.3 (007)22 installed.

Declaration of Transmission Channel

The full syntax of declaration of transmission channel operating according to the TALAS protocol is given below:

logical_name=TALAS, *COMn, baud*

where:

COMn	- number of the serial port to which the TALAS emission computer is connected;
baud	- transmission speed expressed in bauds.

The TALAS driver is loaded as a DLL automatically.

EXAMPLE

```
CHAN1 = TALAS,COM1,9600
```

Addressing the Process Variables

The arrays of data of following categories are read from a TALAS computer:

HM	- current half-minute data,
PI	- current partial integrals,
AI	- current values of half-hour integrals,
HI	- half-hour integrals for a current day,

DIW	- daily distribution of half-hour integrals form a previous day,
YIW	- annual distribution of half-hour integrals from a previous day,
DMV	- distribution of an average daily value from a previous day.

From the arrays above the TALAS driver retrieves process data of the following types:

ATM	- current time (of half-minute data) of the TALAS computer, transferred as a number of seconds in DWORD format;
VAL	- variable value in float format;
STA	- variable status in word format;
OTM	- *work time* of variable counted in tenths of second, transferred in long format;
ITM	- integration period in seconds in short format;
DCL	- classes of classification in word format;
DCT	- classification table for the variable, contents of individual classes in word format, maximal table size – 34 elements,
IVT	- table of half-hour integrals for the variable, element is a structure containing time, value (float) and status (word); maximal table size – 48 elements.

Allowable sets: type-data category together with waited format of an address string are given in the table below, where *nn* signifies the successive number (not identifying) of the variable in the variable list of the TALAS computer in the range <1..128> for HM, PI and AI categories and <1..64> for HI, DIW, YIW and DMV categories (see: the table below).

Table 52. Allowable Sets: Type-Data Category Together with Waited Format of an Address String for the TALAS Driver.

Address state	Contents	Notes
HM.ATM	dword	actual time of TALAS station in [s]
HM.VAL.nn PI.VAL.nn AI.VAL.nn	float	variable value
HM.STA.nn PI.STA.nn AI.STA.nn	word	variable status
HI.VAL.nn	float	half-hour integral of variable
HI.STA.nn	word	status of the above-mentioned integral

HI.IVT.nn	struct {
	struct
	xtime time;
	float value;
	word status;
	}
		table; size 48

DIW.VAL.nn YIW.VAL.nn DMV.VAL.nn	float	average value
DIW.OTM.nn YIW.OTM.nn DMV.OTM.nn	long	working time of variable in [0.1s]
DIW.ITM.nn YIW.ITM.nn DMV.TTM.nn	short	integration period in [s]
DIW.DCL.nn.mm YIW.DCL.nn.mm DMV.DCL.nn.mm	word	mm: class number <1..34>
DIW.DCT.nn YIW.DCT.nn DMV.DCT.nn	word	table; size 34

Driver Configuration

Each defined logical channel has its own section, the name of which must be the same as the name of the logical channel. The items, given below, may be used in sections of logical channels using the TALAS driver.

☑ ***baud=number***

☑ ***bod=number***

☑ ***bps=number***

Meaning	- the item is used to declare a transmission speed. The item value has priority over a transmission speed given in the definition of the logical channel.
Default value	- by default, the transmission speed is assumed to be equal to 9600 Bd.
Defining	- manual.
Parameters:	
number	- transmission speed in bauds.

☑ ***parity=check_type***

Meaning	- the item used to declare a method of the parity check.
Default value	- by default, it is assumed the even parity check.
Defining	- manual.
Parameters:	
check_type	- identifier of the way of parity check:

n	-	no parity bit,
o	-	odd parity check,
e	-	even parity check,
m	-	mark,
s	-	space.

EXAMPLE

```
parity=e
```

☑ ***stop=number***

Meaning	- the item is used to declare a number of stop bits.
Default value	- by default, it is assumed 1 stop bit.
Defining	- manual.
Parameters:	
number	- number of stop bits: 1 or 2.

☑ ***word=number***

☑ ***word_length=number***

Meaning	- the word item is used to declare a number of bits in a transmitted character.
Default value	- by default, it is assumed that the transmitted character has 8 bits.
Defining	- manual.
Parameters:	
number	- number of bits in a character (from 5 to 8).

EXAMPLE

```
word=8
```

☑ ***timeout=number***

Meaning	- the item is used to declare a waiting time for an answer from the TALAS computer.
Default value	- by default, it is assumed 10 seconds.
Defining	- manual.
Parameters:	
number	- waiting time for an answer in seconds.

KM_Interval=number

Interval=number

Meaning	- the item is used to declare a time interval between readings of values of short-time averages and partial integrals from the TALAS computer.
Default value	- by default, it is assumed to be 30 seconds.
Defining	- manual.
Parameters:	
number	- time interval in seconds.

EXAMPLE

```
KM_Interval=30
```

IW_Interval=number

Meaning	- the item used to declare a time interval between readings of values of half-hour integrals and actual integrals from the TALAS computer. By default it is assumed to be 30 minutes.
Default value	- by default, it is assumed to be 30 seconds.
Defining	- manual.
Parameters:	
number	- time interval in minutes.

FD_Interval=number

Meaning	- the item used to declare a time interval between readings of values of distributions from the TALAS computer.
Default value	- by default, it is assumed to be 60 minutes.
Defining	- manual.
Parameters:	
number	- time interval in minutes.

log=name

Meaning	- the item is used to declare a file to which a diagnostic information of the TALAS driver are written. The item is dedicated to test purposes.
Default value	- by default, the file is not created.
Defining	- manual.
Parameters:	
name	- file name.

1.62. CtTwinCAT - Driver of ADS Protocol for TwinCAT System

Driver Use

The CtTwinCAT driver is designed to exchange data between the **asix** system and the TwinCAT system of Beckhoff Industrie Elektronik. The communication between systems is realized through Ethernet in two modes, depending on the firmware a Beckhoff controller is delivered with.

There are two versions of the CtTwinCAT driver:
1. CtTwinCat.dll - uses the TcAdsDll library provided by Beckhoff.
2. CtTwinCatTcpip.dll - uses ADS/AMS over TCPIP interfaces (without Beckhoff libraries).

CtTwinCAT services the following devices:
- controllers of the CX1000 series;
- TwinCAT PLC (PC based control system);
- controllers of the BC9000 series.

Declaration of Transmission Channel

The CtTwinCAT driver is loaded by the universal **asix** driver – UNIDRIVER.

The declaration of transmission channel using CtTwinCat.dll has the following form:

channel_name=UNIDRIVER, CtTwinCAT, Port=*port_nr*; Server=*AMS_address* [;Timeout=*ms_number*] [;TimeSynchr=*addr*][;TimeSynchrPeriod=*period*][; ServiceType=*type*][;StopAlarmNr=*number*]

where:

CtTwinCAT	- driver name;
port_nr	- port number of the TwinCAT system router, for example 801 (RTS1), 811 (RTS2);
AMS_address	- AMS address (4 first elements) of the TwinCAT system router;
ms_number	- optional; timeout (in msec) of the ADS operation; a default value – 50000 ms;
addr	- optional; date & time frame is sent to the PLC at the given address (may be symbolic or absolute);
period	- period (in seconds) between two successive date & time frames sent to the PLC; a default period - 60 seconds;

type - service type: CX,PC-PLC (1) orBC9000 (2); a default value is 1;

StopAlarmNr=*number* - allows alarm generation when PLC switching over the STOP state.

EXAMPLE

Below there is an example of declaration of transmission channel named PLC1, that is used for communication with the TwinCAT system with AMS 10.10.10.254.1.1 and the 801 (RTS1) port with a time synchronization frame sent to the symbolic address *.tabWord*.

PLC1= UNIDRIVER, CTTWINCAT, Port=801; Server=10.10.10.254;
TimeSynchr=.tabWord

The declaration of transmission channel using CtTwinCatTcpip.dll has the following form:

channel_name=UNIDRIVER, CtTwinCATTcpip, Port=*port_nr*;
Server=*AMS_address*, SerwerIP=*controller_IP_address*,
Client=*AmsNetId_address* [;Timeout=*ms_number*] [;TimeSynchr=*addr*]
[;TimeSynchrPeriod=*period*][;ServiceType=*type*][; StopAlarmNr=*number*]

where:

CtTwinCATTcpip - driver name;

port_nr - port number of the TwinCAT system router, for example 801 (RTS1), 811 (RTS2);

AMS_address - AMS address (4 first elements) of the TwinCAT system router;

controller_IP_address - controller IP address;

AmsNetId_address - AmsNetId address that identifies together with an **asix** computer IP a connection **asix** <-> controller; both addresses must have their equivalents in the table of controller project routs to make data exchange possible; it is passed that AmsNetId is the same as an **asix** computer IP (for simplification of connection parameterization);

ms_number - optional; timeout (in msec) of the ADS operation; a default value – 50000 ms;

addr - optional; date & time frame is sent to the PLC at the given address (may be symbolic or absolute);

period - period (in seconds) between two successive date & time frames sent to the PLC; a default period - 60 seconds;

type - service type: CX,PC-PLC (1) orBC9000 (2); a default value is 1;

StopAlarmNr=*number* - allows alarm generation when PLC switching over the STOP state.

EXAMPLE

channel= UNIDRIVER, CTTWINCATTCPIP, Port=801; Server=10.10.10.254;
Client=10.10.10.191; ServiceType=1

The date & time frame structure is given below:

struct dateTime

```
 {
  word wYear;
  word wMonth;
  word wDayOfWeek;
  word wDay;
  word wHour;
  word wMinute;
  word wSecond;
  word wMilliseconds;
  word wSynchr;
 };
```

where:

wYear	- year	1970 ~ 2106
wMonth	- month	1 ~ 12 (January = 1, February = 2, etc)
wDayOfWeek	- day of week	0 ~ 6 (Sunday = 0, Monday = 1, itd.)
wDay	- day of month	1 ~ 31;
wHour	- hour	0 ~ 23;
wMinute	- minute	0 ~ 59;
wSecond	- second	0 ~ 59;
wMilliseconds	- millisecond	0 ~ 999;
wSynchr	- synchro marker	1 is written with the new data & time frame

Types of Process Variable Addresses

Two types of addresses are allowed in the TwinCAT system:

- **symbolic** – variable symbolic address. The form is as follows:

  [<text1>].<text2>

where:

text1, *text2* - ASCII string.

Symbolic addresses of all variables of the project realized in the TwinCAT system are placed in the project directory, in an XML file with the extension *.tpy – addresses are entered in sections identified by the key word *<Symbol>*.

EXAMPLE

.engine
MAIN.devUP

- *direct* – address in the form **Group:Offset**. Both **Group** and **Offset** are declared in HEX format and delimited with a colon (:).

EXAMPLE

F030:03EA

EXAMPLE

Examples of process variable declarations.

```
JJ_00, variable valFloat (REAL),             .valFloat,  PLC1, 1, 1, NOTHING_FP
JJ_01, variable valWord (WORD),              .valWord,  PLC1, 1, 1, NOTHING
JJ_02, variable MAIN.engine (BOOL),          MAIN.engine,      PLC1, 1, 1,
                                             NOTHING_BYTE
JJ_03, variable with address F030:5 (BYTE), F030:5, PLC1, 1, 1, NOTHING_BYTE
```

Variable Type

For variables with a symbolic address the raw type of a variable is taken from the variable's type declared in the TwinCAT system. This type is converted into the type required by the user.

For variables with a direct address the raw type of a variable is the real type of conversion function defined for the variable. Based on this type:

- the appropriate number of bytes are read/written from/into the address specified by **Group:Offset**;
- the conversion of read bytes, according to the type defined for the variable, is performed.

Data & Time Stamp

Data read from the driver is stamped with the PC local date & time, evaluated at the moment when the ADS request is completed.

Driver Configuration

The driver configuration is performed by using the separate section named **[CTTWINCAT]** – for CtTwinCat.dll or **[CTTWINCATTCPIP]** – for CtTwinCatTcpip.dll. By means of this section it is possible to declare:

- log file and its size,
- size of the buffer used for ADS requests,
- log of telegrams.

☑ ***LOG_FILE=file_name***

Meaning	- the item allows to define a file to which all diagnostic messages of the driver and the information about ADS requests will be written. If the item doesn't define a full path, the log file will be created in the current directory. The log file should be used only during the **asix** system start-up.
Default value	- by default, the log file is not created.
Parametr:	
file_name	- name of the log file.

☑ ***LOG_FILE_SIZE=number***

Meaning	- the item allows to define the size of the log file.
Default value	- by default, the item assumes that the log file has a size of 1 MB.
Parameter:	
number	- size of the log file in MB.

☑ ***MAX_BUFFER_LEN=number***

Meaning	- the item allows to define the size of the buffer used to perform ADS read/write requests.
Default value	- the default size of the buffer is 1800 bytes.
Parameter:	
number	- size of the buffer in bytes.

☑ ***LOG_OF_TELEGRAMS=YES/NO***

Meaning	- the item allows to write the info about performed ADS requests to the log file. This option should be used only during the **asix** system start-up.
Default value	- by default, the driver does not write the info about ADS requests to the log file.

1.63. ZdarzenieZmienna Driver

Driver Use

The ZdarzenieZmienna driver is used to generate process variables of WORD types (16-bit word) on the basis of current values of alarm events in the **asix** system.

Declaration of Transmission Channel

The ZdarzenieZmienna driver is a dynamic library DLL with an interface meeting requirements of the ASMEN module. ASMEN activates the driver after having found in the application INI file, in the ASMEN section a channel definition calling to the ZdarzenieZmienna driver and having the following form:

channel_name = UniDriver, ZdarzenieZmienna

where:

channel_name	- channel name of the Asmen module;
UniDriver	- module intermediate between the ZdarzenieZmienna driver and the ASMEN module.

After starting the **asix** system, the ZdarzenieZmienna driver receives from ASMEN one after the other an algorithm, taken from the definition file, calculating the value of each process variable. This algorithm is executed at each reading the variable and its result becomes a value of the process variable.

Variables supplied by the driver are variables only for reading.

Calculating the asix Process Variables

The alarm module of the **asix** system generates values of individual alarm events, which are identified by its number in the file of alarm definitions.

Two ways of determining values of alarm event exist:

- alarm event assumes a value of 1 when the alarm is active, 0 when it is inactive.
- alarm event assumes a value of 1 when the alarm is active and not acknowledged, 0 when it is active and acknowledged or inactive.

Alarm events are used to calculate **asix** process variables identified by means of their name in the whole system.

Definition of ASMEN variables (in database of variables) contains the way of calculating values of an individual variable of the **asix** system.

The declaration of each variable has the following form:

variable_name,variable_description,algorithm for calculating values,channel,1,frequency,NOTHING

A detailed description of components of this line may be found in chapter Declaring process variables.

The algorithm for calculating values is specific for this described driver. There are three algorithms of operations on alarm events that generate a value of the process variable:
- composition of alarm events,
- logical sum of alarm events,
- logical product of alarm events.

An alarm event is treated as a number of alarm in the alarm file with a suffix N. The N suffix signifies that an alarm event assumes a value of 1 only when the alarm with a given number is active and not acknowledged.

<u>Composition of Alarm Events</u>

The algorithm form:

Composition:<Alarm event >; <Alarm event >; ...

or

MERGE:<Alarm event >; <Alarm event >; ...

The interpretation result of this algorithm is a number, the bit no. 0 of which has a value equal to the first alarm variable, bit no. 1 equal to the second alarm variable etc. The algorithm may contain maximally 16 alarm variables.

EXAMPLE

MERGE: 11; 12; 13N

In the algorithm, the alarm variables may be omitted and then the state of bit corresponding with omitted alarm variable is always equal to 0.

EXAMPLE

MERGE: 11; ; 13; ;15; 16
The bits no. 1 and 3 have always a value of zero.

<u>Logical Sum of Alarm Events</u>

The algorithm form:

Logical_sum:<Alarm event>; <Alarm event>; ...

or

OR:<Alarm event >; <Alarm event >; ...

The interpretation result of this algorithm is a value of 1 if at least one of the alarm variables has a value of 1, 0 – otherwise. The algorithm may contain maximally 16 alarm variables.

EXAMPLE

OR: 11N; 12; 13

<u>Logical Product of Alarm Events</u>

The algorithm form:

LogicalProduct:<Alarm event>; <Alarm event>; ...

or

AND:<Alarm event>; <Alarm event>; ...

The interpretation result of this algorithm is a value of 1 if all alarm variables have a value of 1, 0 - otherwise. The algorithm may contain maximally 16 alarm variables.

EXAMPLE

AND: 11; 12N; 13N

<u>Subjection of Values of Alarm Events to Value Status</u>

Three algorithm above may also use alarm events, the value of which, taken from the alarm module, is still corrected by the state of the other alarm event treated as a status. You should use then the writing: alarm event/status e.g.101/102. When the status is equal to 0, then it does not change the value of alarm event. Otherwise a measure error for all **asix** process variables is generated. It is calculated on the basis of this status.

EXAMPLE

AND: 11/110; 12; 13
or
OR: 11; 12/120; 13/130

EXAMPLE OF USING THE DRIVER

Our task is to cause a text on synoptic diagram to blink while simultaneous exceeding limits for two temperatures T1 and T2. Two-state signals about exceeding appropriate temperatures we obtain from the object. We assume that alarms in the system are already activated and a file of alarms definitions exists.

Declarations in the Configuration Application File

In the configuration file for the application we place a declaration initiating the driver ZdarzenieZmienna for the channel named EVENT:

```
EVENT=UniDriver,ZdarzenieZmienna
```

Complement of File of Alarms Definitions

In the alarm file you should add two alarms, which signals exceeding the temperatures T1 and T2. For example, we place them under numbers: 101 for exceeding the temperature T1 and 102 for exceeding the temperature T2:

```
101,AL,Exceeding of temperature T1
102,AL,Exceeding of temperature T2
```

Creation of File Defining the Asmen Variables

You should declare new variable by using *VariableBase Manager*.

The process variable should assume a value of 1 when both alarms 101 and 102 are active (they assume a value of 1), otherwise the value of process variable should be equal 0. Let's name it T1_T2_MAX.

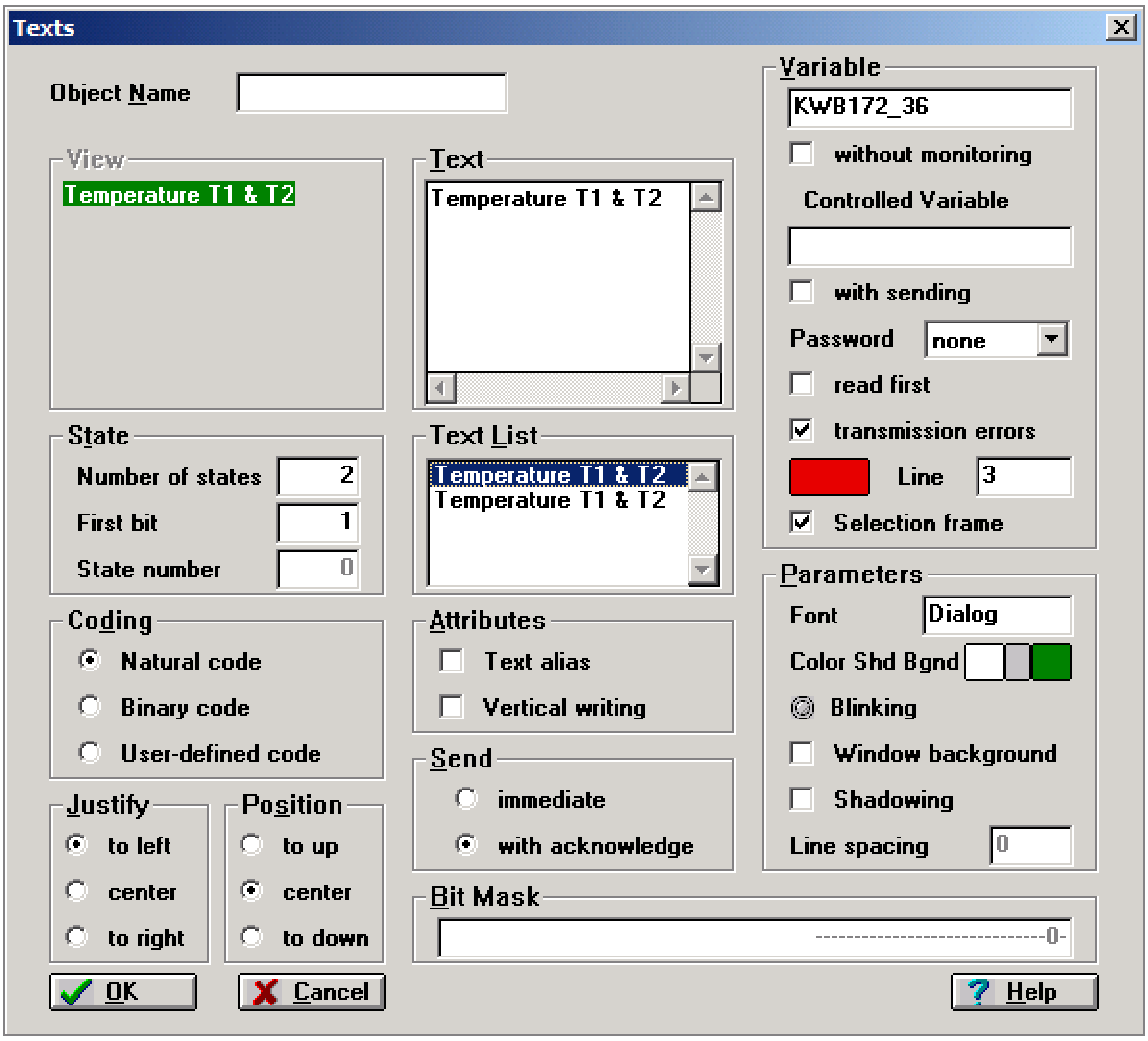
The file of the variable definition should contain the following line:

```
T1_T2_MAX,Process variable of exceeding T1 and
T2,AND:101;102,EVENT,1,1,NOTHING
```

Location of a Text Object on the Application Synoptic Diagram

After having opened the mask under *Constructor* you should place on it the TEXT object and parameterize it on our variable T1_T2_MAX. The object must have two states. For the value of a monitored variable 0 the text "Temperatures T1 and T2" are displayed. For the value 1 a text "Exceeding of temperatures T1 and T2" with a blink attribute.

Window with object configuration.


Texts

Object Name:

View: Temperature T1 & T2

Text: Temperature T1 & T2

Variable: KWB172_36
- [ ] without monitoring

Controlled Variable:
- [ ] with sending

Password: none
- [ ] read first
- [✓] transmission errors

🟥 Line: 3
- [✓] Selection frame

State
- Number of states: 2
- First bit: 1
- State number: 0

Text List: Temperature T1 & T2; Temperature T1 & T2

Parameters
- Font: Dialog
- Color Shd Bgnd: ⬜ ▨ 🟩
- Blinking
- [ ] Window background
- [ ] Shadowing
- Line spacing: 0

Coding: [✓ Natural code] Binary code User-defined code

Attributes
- [ ] Text alias
- [ ] Vertical writing

Send: immediate [✓ with acknowledge]

Justify: [✓ to left] center to right

Position: to up [✓ center] to down

Bit Mask: --------------------------------0-

OK Cancel Help

Figure 5. Window with OBJECT Parameterization.

1.64. CtZxD400 - Driver of Protocol of Landys & Gyr ZxD400 Electric Energy Counters

Driver Use

The CtZxD400 driver is used for data exchange between **asix** and electric energy counters of ZxD400 type manufactured by Landys & Gyr, via the RS-485 interface. The driver is not adapted for data exchange through an optical connection, because it demands the protocol with initial negotiations of transmission speed to be used.

The driver allows readout of register statement of a counter as well as registration of data (read by commands *Read Log Book* or *Read Load Profile*) in files.

Declaration of Transmission Channel

The syntax of declaration of transmission channel using the driver is as follows:

```
Channel=UNIDRIVER, CtZxD400,Port=number
[;Baudrate=number][;Period=number]
```

where:

UNIDRIVER	- name of the universal **asix** driver UNIDRIVER;
CtZxD400	- name of the driver used for communication with the counter;
Port	- number of the COM serial port;
Boudrate	- speed of transmission between computer and device; the following speeds are acceptable: 1200,2400, 4800, 9600, 19200, 38400 Bd; a default value is 2400 Bd;
Period	- timeout (in seconds) between successive readouts of counter registers. A default value is 10 seconds.

EXAMPLE

An exemplary channel declaration on the COM2 port:

CHANNEL = UNIDRIVER, CtZxD400, Port=2; Period=20; Baudrate=9600

Declaring the Process Variables

The syntax of the symbolic address of the process variable is as follows:

"[*counter_name*]/*register_code*"

where:

counter_name	- (option); defines the controller unique name used in multipoint installations to identify particular controllers;
counter_name	- it corresponds to *Device address* according to PN-EN 61107; *counter_name* may be omitted in point-point connection;
register_code	- code and index of the counter register compatible with a readout list - that is loaded into the counter by the manufacturer in the parameterization stage.

NOTICE *All the variable values are of FLOAT type.*

EXAMPLE

```
/* C.1.0          - identification number of the counter */
JJ_01, identification number of the counter, "/C.1.0",CHANNEL, 1, 1, NOTHING_FP

/* 1.8.0          - register of consumed active energy*/
JJ_03, register of consumed active energy, "/1.8.0",CHANNEL, 1, 1, NOTHING_FP

/* C.8.0          - total worktime */
JJ_03, total worktime, "/C.8.0",CHANNEL, 1, 1, NOTHING _FP
```

Driver Parameterization

Driver parameterization takes place with use of the separate section named **[CTZxD400]**. Using this section, you may declare:

- log file,
- log file size,
- log of telegram.

log_file_name

Meaning	- allows to define a file to which all the diagnostic messages of the driver will be written.
Default value	- by default, the log file is not created.
Defining	- manual.

LOG_FILE_SIZE=number

Meaning	- this item is used to define the size of the log file defined with use of the LOG_FILE item.
Default value	- by default, the log file size is 1 MB.

Parameter:	
number	- log file size in MB.
Defining	- manual.

☑ ***LOG_OF_TELEGRAMS =YES | NO***

Meaning	- this item allows contents of telegrams transferred between driver and controllers to be written into the log file (declared with use of the LOG_FILE item). The referred item should only be used in the **asix** system start-up.
Default value	- by default, value of this item is set to NO.
Defining	- manual.

Parameterization of Particular Counters

The dirver allows the set of individual parameters concerning service of particular counters to be transferred in separate sections. The name of such section is composite of the following elements:

channel_name:*counter_name*

where:

channel_name	- ASMEN's channel name in which the given counter is serviced;
counter_name	- address name of the counter (the name used in a variable address); *counter_name* may be empty, if point-point installation is used.

EXAMPLE 1

ASMEN's channel name	CHANNEL
Counter name	counter1
Section name	CHANNEL:counter1

EXAMPLE 2

ASMEN's channel name	CHANNEL
Counter name	is not used
Section name	CHANNEL:

A parameterization of the counter may be performed by the following items:
- time maker,
- log book file,
- log book file size,
- log book read period,
- log profile file,
- log profile file size,
- log profile read period.

☑ ***TIME_MAKER = register_code [,register_code]***

Meaning	- it allows to define a register (or two registers), that includes the data and time stamp transmitted from the counter. It is assumed that, if one register is declared, it contains data and time in the 'YY-MM-DD hh:mm:ss' format. If the couple of registers is declared, it is assumed

that the first register contains data in the 'YY-MM-DD' format, and the second one contains time in the 'hh:mm:ss' format.

Default value - by default, it is assumed that the PC's data and time stamp from the moment of the end of data receiving will be assigned to variables transmitted from the counter.

☑ ***LOG_BOOK_FILE =log_file_name***

Meaning - events read from the counter are written to a text file in 'csv' format. Each event is written in a separate line. Events in the file are ordered according to the growing time stamps. An exemplary event form is as follows:

```
P.98;1;2005-05-10
08:14:33;0048;;3;;;F.F;;1.8.0;kWh;024;00000000;0000.
0000
```

LOG_BOOK_FILE allows to define a file path in which the book file will be written.

Default value - by default, the log book file is not created.

☑ ***LOG_BOOK_FILE_SIZE =number***

Meaning - this item is used to define the size of the log file defined with use of the LOG_BOOK_FILE item.

Deafult value - by default, the log file size is 10 MB.

Parameter:

number - log file size in MB.

☑ ***LOG_BOOK_READ_PERIOD =number***

Meaning - it allows to determine the cycle of the event log readout from the counter.

Default value - by default, the event log readout takes place every hour.

Parameter:

number - cycle of the event log readout (in hours).

☑ ***LOG_PROFILE_FILE =log_file_name***

Meaning - profiles read from the counter are entered to a text file in 'csv' format. Each profile is written in a separate line. Profiles in the file are ordered according to the growing time stamps. An exemplary profile form is as follows (the exemplary profile form is broken into two lines to be more clear):

```
P.01;1;2005-05-10 12:45:00;0008;15;6;1.5.0;kW;5.5.0;kvar;8.5.0;kvar;
32.7;V;52.7;V;72.7;V;0.0000;0.0000;0.0000;---.-;---.-;---.-
```

LOG_PROFILE_FILE allows to define a file path in which the profile file will be written.

Default value - by default, the profile log file is not created.

☑ *LOG_PROFILE_FILE_SIZE =number*

Meaning	- this item is used to define the log file size, defined with use of the LOG_PROFILE_FILE item.
Default value	- by default, the log file size is 10 MB.
Parameter:	
number	- log file size in MB.
Defining	- manual.

☑ *LOG_PROFILE_READ_PERIOD =number*

Meaning	- it allows to determine the cycle of profile log readout from the counter.
Default value	- by default, profile log readout takes place every hour.
Parameter:	
number	- cycle of profile log readout (in hours).

☑ *LOG_BOOK_DATA =YES/NO*

Meaning	- it allows to declare whether detailed description of parsing of particular event log lines should be entered to the driver log. The log file should be used only while the **asix** system start-up.
Default value	- by default, event parsing details are not written to the driver log.

☑ *LOG_PROFILE_DATA =YES/NO*

Meaning	- it allows to declare whether detailed description of parsing of particular profile log lines should be entered to the driver log. The log file should be used only while the **asix** system start-up.
Default value	- by default, event parsing details are not written to the driver log.

☑ *METER_DATA =YES/NO*

Meaning	- it allows to declare whether detailed description of parsing of particular data (read from the counter) lines should be entered to the driver log. The log file should be used only while the **asix** system start-up.
Default value	- by default, readout data parsing details are not written to the driver log.

☑ *HISTORY_SCOPE =number*

Meaning	- it allows to declare period of profile and event history for each counter individually.
Default value	- 0, that means reading since 00:00 of a current day.
Parameter:	
number	- time passed in days.

EXAMPLE

There is an exemplary section describing the counter in the CHANNEL channel below. Because the counter has not declared name (*counter_name*), the character ":" ends the name of the section.

```
[CHANNEL:]
TIME_MARKER =  0.9.2,  0.9.1
LOG_BOOK_FILE = c:\tmp\ctZxD400\book.log
LOG_BOOK_FILE_SIZE  = 4
LOG_BOOK_READ_PERIOD = 1
LOG_PROFILE_FILE = c:\tmp\ctZxD400\profile.log
LOG_PROFILE_FILE_SIZE = 2
LOG_PROFILE_READ_PERIOD = 1
LOG_BOOK_DATA = YES
LOG_PROFILE_DATA = YES
```

EXAMPLE

An exemplary section of the driver.

```
[CTZXD400]
LOG_FILE =d:\tmp\CtZxD400\ak.log
LOG_FILE_SIZE =3
LOG_OF_TELEGRAMS =YES
```

INDEX

List of Figures

List of Tables