



IEC 61131-3 Programming (LogicLab)

ELSIST S.r.l.
Electronic systems

Via G. Brodolini, 15 (Z.I.)
15033 CASALE M.TO
ITALY

Internet: http://www.elsist.it
Email: elsist@elsist.it

TEL. (39)-0142-451987
FAX (39)-0142-451988

INDEX

1 - LogicLab.....	7
2 - System resources.....	8
2.1 - Memory structure.....	9
2.2 - Backup memory (Retain).....	10
2.3 - Memory access.....	11
3 - Data types definition.....	12
3.1 - FILEP, file pointer.....	12
3.2 - SYSSERIALMODE, serial port communication mode.....	12
3.3 - SYSCANMESSAGE, CAN message.....	12
4 - System variables.....	13
4.1 - Read only variables (System variables).....	14
4.2 - Read and write variables (System variables).....	16
4.3 - Product unique ID.....	17
5 - Data definition.....	18
5.1 - Variable types definition.....	18
5.2 - Task ID definition.....	18
5.3 - TermIO, definitions for terminal I/O.....	19
5.4 - FSeek origin definition.....	19
5.5 - Serial mode definition.....	19
5.6 - CAN bit rate definition.....	19
5.7 - Digital input mode.....	20
5.8 - Digital output mode.....	20
5.9 - Analog to digital mode.....	21
5.10 - Digital to analog mode.....	22
5.11 - Spy mode, spy mode definition.....	22
6 - Functions defined by LogicLab.....	23
6.1 - Mathematical and trigonometric functions.....	24
6.2 - String functions.....	26
7 - Functions and FBs.....	27
- Functions.....	27
- Function Blocks.....	27
7.0.1 - Functions and embedded FBs.....	27
7.0.2 - Libraries.....	28
7.0.3 - Import library.....	28
7.0.4 - Import a library.....	29
7.0.5 - Import library objects.....	30
7.0.6 - Considerations about links to library and import of objects.....	31
7.0.7 - Functions and function blocks protection.....	32
7.1 - Functions and FBs for Flip/Flop management.....	33
7.1.1 - F_TRIGGER, Falling edge trigger.....	33

7.1.2 - R_TRIGGER, Raising edge trigger.....	34
7.1.3 - RS, Reset/Set flip flop.....	35
7.1.4 - SR, Set/Reset flip flop.....	36
7.2 - Functions and FBs for timers management.....	37
7.2.1 - eTOF, Timer Off.....	37
7.2.2 - eTON, Timer On.....	38
7.2.3 - eTP, Timer pulse.....	39
7.3 - Functions and FBs for counters management.....	40
7.3.1 - CTD, Counter Down.....	40
7.3.2 - CTU, Counter Up.....	42
7.3.3 - CTUD, Counter Up/Down.....	43
7.4 - Functions and FBs for data conversion.....	45
7.4.1 - VBitTest, Variable bit test.....	45
7.4.2 - VBitSet, Variable bit set.....	46
7.4.3 - BitToByte, Bit to byte conversion.....	47
7.4.4 - ByteToBit, Byte to bit conversion.....	49
7.4.5 - ByteToWord, Byte to word conversion.....	50
7.4.6 - WordToByte, Word to byte conversion.....	51
7.4.7 - DoubleToWord, Double to word conversion.....	52
7.4.8 - Word.ToDouble, Word to double conversion.....	53
7.4.9 - ToLower, Uppercase to lowercase letter conversion.....	54
7.4.10 - ToUpper, Lowercase to uppercase letter conversion.....	55
7.4.11 - LEArrayToVar, Little endian array to variable conversion.....	56
7.4.12 - BEArrayToVar, Big endian array to variable conversion.....	57
7.4.13 - VarToLEArray, variable to little endian array conversion.....	58
7.4.14 - VarToBEArray, variable to big endian array conversion.....	59
7.5 - Functions and FBs of system utility.....	60
7.5.1 - SysGetSysTime, get system time.....	60
7.5.2 - SysSetTaskLpTime, set task loop time.....	62
7.5.3 - SysGetRandom, get random number.....	63
7.5.4 - SysGetLastError, get last error.....	64
7.5.5 - SysPCodeAccept, accepts the protection code.....	65
7.5.6 - SysCalcCrc, CRC calculation.....	66
7.5.7 - SysMAlloc, memory allocation.....	67
7.5.8 - SysRMAalloc, relocatable memory allocation.....	68
7.5.9 - SysRMFree, relocatable memory free.....	69
7.5.10 - SysGetEndianness, get the system endianness.....	70
7.5.11 - SysGetUTCDateTime, get the system Date/Time on UTC.....	71
7.5.12 - SysSetUTCDateTime, set the system Date/Time on UTC.....	72
7.5.13 - SysTimeZoneAdj, adjust date and time.....	73
7.5.14 - SysSpyData, system spy data.....	74
7.6 - Functions and FBs for Date/Time management.....	76
7.6.1 - SysETimeToDate, epoch time to date conversion.....	76
7.6.2 - SysDateToETime, date to epoch time conversion.....	78
7.7 - Functions and FBs for I/O terminal management.....	80
7.7.1 - Sysfopen, file open.....	80
7.7.2 - SysFlsOpen, get the file open status.....	82
7.7.3 - Sysfclose, file close.....	83
7.7.4 - Sysfgetc, get character from file.....	84

7.7.5 - Sysputc, put character to file.....	85
7.7.6 - Sysread, read data from file.....	87
7.7.7 - Sysfwrite, write data to file.....	88
7.7.8 - SysFGetChars, get input available characters from file.....	89
7.7.9 - SysFGetOSpace, get output available space on file.....	90
7.7.10 - SysFGetBfSize, get file Rx input buffer size.....	91
7.7.11 - SysFGetObfSize, get file Tx output buffer size.....	92
7.7.12 - SysFIBfClear, file input buffer clear.....	93
7.7.13 - SysFOBfClear, file output buffer clear.....	94
7.7.14 - SysFOBfFlush, file output buffer flush.....	95
7.7.15 - SysVarprintf, variable print to file.....	96
7.8 - File system.....	97
7.8.1 - Sysremove, file remove.....	98
7.8.2 - Sysrename, file rename.....	99
7.8.3 - Sysfilelength, file length.....	100
7.8.4 - Sysseek, file seek.....	101
7.8.5 - SysDirListing, directory listing.....	102
7.9 - Functions and FBs for serial port management.....	104
7.9.1 - SysSerialPort, manage serial port.....	105
7.9.2 - SysGetSerialMode, get serial mode.....	107
7.9.3 - SysGetSerialCTS, get serial CTS signal status.....	109
7.9.4 - SysSetSerialDTR, set DTR signal status.....	110
7.10 - Functions and FBs for CAN bus management.....	111
7.10.1 - SysCANSetMode, set the CAN controller mode.....	112
7.10.2 - SysIsCANRxTxAv, checks if CAN Rx or Tx is available.....	113
7.10.3 - SysCANRxMsg, receives a CAN message.....	114
7.10.4 - SysCANTxMsg, transmit a CAN message.....	115
7.11 - Functions and FBs for string management.....	116
7.11.1 - eLEN, string length.....	116
7.11.2 - eFIND, string find.....	117
7.11.3 - MemSet, memory set.....	118
7.11.4 - MemCopy, memory copy.....	119
7.11.5 - SysVarsprintf, variable print to string.....	120
7.11.6 - SysLWVarsprintf, variable print to string with append.....	122
7.11.7 - SysVarsscanf, extracts values from string.....	123
7.11.8 - SysFWVarsscanf, extracts values from string with find.....	124
7.12 - Functions and FBs for extension modules management.....	125
7.12.1 - SysPhrInfos, get infos from peripheral modules.....	125
7.12.2 - SysGetPhrDI, get peripheral digital input.....	126
7.12.3 - SysSetPhrDO, set peripheral digital output.....	129
7.12.4 - SysGetAnInp, get analog input.....	131
7.12.5 - SysSetAnOut, set analog output.....	133
7.12.6 - SysGetCounter, get counter.....	135
7.12.7 - SysGetEncoder, get encoder input.....	137
7.12.8 - SysSetPWMOut, set PWM output.....	139
7.12.9 - SysPhrVRd, read variable from peripheral module.....	141
7.12.10 - SysPhrVWr, write variable to peripheral module.....	142
7.12.11 - SysI2CWrd, writes/reads on I2C extension bus.....	143
7.12.12 - StrainGaugeAcq, strain gauge acquisition.....	144

7.13 - Functions and FBs of general utility.....	146
7.13.1 - BLINK, blink command.....	146
7.13.2 - BlinkValue, blink out value.....	147
7.13.3 - ModbusMaster, modbus master.....	148
7.13.4 - ModbusSlave, modbus slave.....	151
7.13.5 - OnOffCycle_v1, on/off cycle with random times.....	154
7.13.6 - PIDMng, PID management.....	156
7.13.7 - PWMOut, PWM output management.....	158
7.13.8 - SysDMXMng, DMX management.....	159
7.13.9 - IOEncoder, incremental encoder over I/O.....	161
7.13.10 - GetISO1155Crc, calculate CRC according ISO1155.....	162
7.13.11 - IODataExchange, exchange data by using logic I/O.....	163
7.13.12 - Average, value average.....	165
7.13.13 - HIDClkDtaReader, HID RFID clock/data reader.....	166
7.13.14 - Linearize, linearize a non linear value.....	168
7.13.15 - ValueScale, scales a value.....	169
7.13.16 - GetPolynomialCRC, polynomial CRC calculation.....	170
7.13.17 - LRamp, linear ramp.....	171
7.13.18 - VaPotentiometer, voltage acquisition potentiometer.....	172
7.13.19 - ResistorValue, resistor value acquire.....	173
7.14 - DLMS protocol, or IEC 62056-21.....	174
7.14.1 - IEC62056_21Rd, IEC62056-21 protocol read.....	175
7.15 - Functions and FBs for modem management (eModemLib_E000).....	177
7.15.1 - ModemCore_v3, modem core management.....	178
7.15.2 - ModemSMSReceive, receive a SMS message.....	180
7.15.3 - ModemSMSRxCmd_v1, receive a SMS command.....	181
7.15.4 - ModemSMSSend_v2, send a SMS message.....	182
7.15.5 - ModemPhoneCall_v1, executes a phone call.....	185
7.15.6 - ModemHTTPGet, executes a HTTP Get request.....	186
7.16 - Functions and FBs for One-Wire management (ePLC1WireLib_C000).....	189
7.16.1 - sOWireMng, One-Wire management.....	190
7.16.2 - sOWRdIdentifier, One-Wire read ROM identifier.....	191
7.16.3 - sOWRdTTemperature, One-Wire read temperature.....	193
7.16.4 - sOWRdHumidity, One-Wire read humidity.....	195
7.17 - Functions and FBs for networking management.....	197
7.17.1 - SysTCPServer, accepts TCP/IP connections.....	198
7.17.2 - SysTCPClient, opens a TCP/IP connection.....	200
7.17.3 - SysUDPServer, accepts UDP connections.....	202
7.17.4 - SysUDPClient, opens a UDP connection.....	204
7.17.5 - SysGetIpInfos, returns IP infos.....	206
7.17.6 - SysIPReach, IP address is reachable.....	207
7.17.7 - UDPDataTxfer, UDP data transfer.....	208
7.17.8 - DataStreamExch, exchanges data between two I/O streams.....	210
7.17.9 - ModbusTCPGateway, modbus TCP gateway.....	212
7.17.10 - SNTPRequest, sends a SNTP request.....	214
7.17.11 - DNSRequest, sends a DNS request.....	216
7.17.12 - HTTPProtocol, HTTP protocol management.....	218
7.18 - Functions and FBs for Hw Group products (eHwGSpLib).....	223
7.18.1 - STESnmpAcq, STE thermometer acquisition over SNMP.....	224

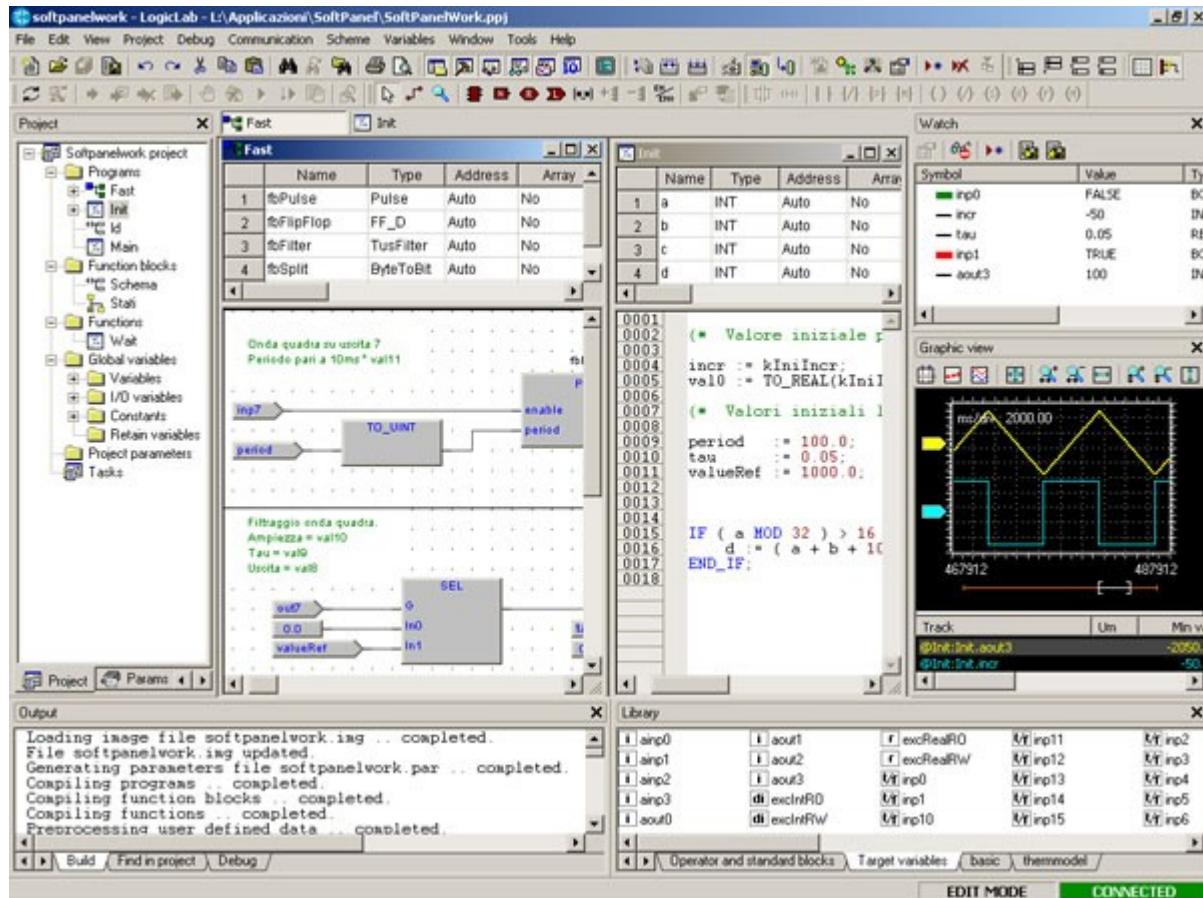
7.18.2 - sHWgSProtocol, HW group serial protocol.....	226
7.19 - Functions and FBs for NMEA protocol (eNMEALib).....	228
7.19.1 - NMEASInterface, NMEA system interface.....	229
7.19.2 - GLLSentence, Geographic Position sentence.....	230
7.19.3 - MWVSentence, Wind Speed and Angle sentence.....	232
7.20 - Functions and FBs for Power One inverter (ePowerOneLib).....	233
7.20.1 - AuroraDSPMeasure, Aurora measure request to DSP.....	234
7.20.2 - AuroraCEnergy, Aurora cumulated energy reading.....	237
7.21 - Functions and FB to manage logs (eLogLib).....	239
- Sends notifications to a Syslog server.....	239
7.21.1 - SysLogReport, send a report to Syslog server.....	241
7.21.2 - StringToFile, store string to a log file.....	243
7.21.3 - FileMemoryDump, dump memory on file.....	245
7.22 - Functions and FBs for multimaster communication (eMMasterDTxferLib).....	247
7.22.1 - MMasterDataTxfer, multimaster data transfer.....	249
7.22.2 - DataTxferClient, Data transfer client.....	250
7.22.3 - BroadcastDataSend, broadcast data send.....	252
7.23 - Human machine interface built-in library (eHMBuiltInLib).....	253
7.23.1 - HMIBuiltInMessages, HMI built in messages.....	254
7.23.2 - HMIBuiltInNetlog, Netlog HMI management.....	255
7.23.3 - HMIPicoface, Picoface HMI management.....	257
7.24 - Cctalk protocol management library (eCCTalkProtoLib).....	259
7.24.1 - ccTalkProtocol, manages ccTalk protocol.....	260
7.24.2 - AlbericiAL66, Alberici AL66 coin acceptor.....	261
8 - Communication protocols.....	264
8.1 - Modbus protocol.....	264
8.1.1 - Access to variables from modbus.....	264
8.1.2 - Reading variables from modbus.....	264
8.1.3 - Writing variables from modbus.....	265
8.1.4 - Access to Real time clock from modbus.....	266
8.1.5 - Reading RTC from modbus.....	266
8.1.6 - Writing RTC from modbus.....	266
8.1.7 - Epoch time access from modbus.....	268
8.1.8 - Reading Epoch time from modbus.....	268
8.1.9 - Writing Epoch time from modbus.....	268
9 - How to create user web pages.....	269
9.1 - Web pages criteria.....	270
9.2 - Dynamic web pages.....	271
9.3 - TAGs format.....	272
9.3.1 - Format field.....	272
9.3.2 - Type field.....	273
9.3.3 - Address field.....	273
9.3.4 - Some TAGs example.....	273
9.4 - ARGs format.....	274
9.4.1 - ARG id.....	274
9.5 - Some examples.....	275
9.6 - LogicIO, logic I/O management.....	275
9.7 - Updating pages with AJAX.....	276
10 - Tips and tricks.....	278
10.1 - DWORD variable swap.....	278

10.2 - WORD variable swap.....	278
10.3 - BYTE variable swap.....	279
10.4 - Expand DWORD to 32 BOOL.....	280
10.5 - Compress 32 BOOL into a DWORD.....	281
10.6 - Define non-printable ascii characters.....	282
10.7 - Rx/Tx data to stream.....	283
10.8 - Data types conversion.....	284
10.9 - User Informations and Settings.....	286
11 - Programming examples.....	287
11.1 - Example library.....	287
11.2 - Logic I/O definitions in the examples.....	288
11.3 - Examples provided with LogicLab.....	289
11.3.1 - Example programs list.....	290
12 - Appendix.....	291
12.1 - IL instructions table.....	291
12.2 - ST language operators.....	292
12.3 - ST language statements.....	293
12.4 - Object IDs.....	294
12.5 - Ascii codes table.....	297
12.5.1 - ASCII standard codes table.....	297
12.5.2 - ASCII extended codes table.....	298

DEPRECATE

1 LogicLab

LogicLab is a IEC61131-3 PLC compiler that supports all 5 programming languages of the standard, Easy-to-use textual and graphic editors, extended drag & drop functions applying to all contexts of the framework, various integrated textual and graphic debuggers make LogicLab an efficient development environment and a particularly user-friendly program



The LogicLab's compiler generates directly the machine code for the target processor, guaranteeing high speed execution time. The tool has been developed by Axel, an Italian Company with multiyear experience in industrial automation software production, and customized to be used with our programmable controllers.

2 System resources

The logic I/O are automatically handled by the operating system in the process image. The operating system transfers to the state of all logic inputs in the input image in system memory and transfers the value in the output image from memory system to the logic outputs.

So testing the state of the memory image of the logic inputs, will be checked the status of the logic inputs. (Example **IX0.0** corresponds to the input 0 of module extension 0; **IX1.5** corresponds to the input 5 of the module extension 1).

Writing the image memory of the logic outputs, will be set the status of the logic output (Example **QX0.0** is of the output 0 od extension module 0; **QX1.5** is the logic output 5 of extension module 1).

The **logic I/O** can be managed through the functions [**SysGetPhrDI**](#) and [**SysSetPhrDO**](#).

The **analog I/O** can be managed by [**SysGetAnInp**](#) and [**SysSetAnOut**](#) functions.

The **counters** can be managed by the function block [**SysGetCounter**](#)

The **encoder** inputs can be managed by the function block [**SysGetEncoder**](#).

The **CAN bus** is managed by the function block [**SysCANRxMsg**](#) e [**SysCANTxMsg**](#).

To access to serial ports on the CPU module, the [**Sysfopen**](#) function must used defining the name of the port to use. There are extension modules that are equipped with serial ports, the access to these ports is exactly the same as the ports on the CPU module. Using the definition **PCOMx.y** with **x** indicates the module address and **y** is the port number on the module. (Example **PCOM0.0** 0 defines the port 0 on the module 0; **PCOM1.2** indicates the port 2 on module 1 and so on).

To access the file system (Both on the internal disk od on the SD Card), you must use the [**Sysfopen**](#).

In models with an Ethernet port, you can manage UDP and TCP/IP connections by using the [**Sysfopen**](#).

2.1 Memory structure

The memory of the system is so organized:

DB	Dimensione	Descrizione
IX0	32 Bytes	Logic input module 00 (R)
IX1	32 Bytes	Logic input module 01 (R)
IX2	32 Bytes	Logic input module 02 (R)
IX3	32 Bytes	Logic input module 03 (R)
IX4	32 Bytes	Logic input module 04 (R)
IX5	32 Bytes	Logic input module 05 (R)
IX6	32 Bytes	Logic input module 06 (R)
IX7	32 Bytes	Logic input module 07 (R)
IX8	32 Bytes	Logic input module 08 (R)
IX9	32 Bytes	Logic input module 09 (R)
IX10	32 Bytes	Logic input module 10 (R)
IX11	32 Bytes	Logic input module 11 (R)
IX12	32 Bytes	Logic input module 12 (R)
IX13	32 Bytes	Logic input module 13 (R)
IX14	32 Bytes	Logic input module 14 (R)
IX15	32 Bytes	Logic input module 15 (R)
IX255	32 Bytes	Logic input CPU module (R)
QX0	32 Bytes	Logic output module 00 (R/W)
QX1	32 Bytes	Logic output module 01 (R/W)
QX2	32 Bytes	Logic output module 02 (R/W)
QX3	32 Bytes	Logic output module 03 (R/W)
QX4	32 Bytes	Logic output module 04 (R/W)
QX5	32 Bytes	Logic output module 05 (R/W)
QX6	32 Bytes	Logic output module 06 (R/W)
QX7	32 Bytes	Logic output module 07 (R/W)
QX8	32 Bytes	Logic output module 08 (R/W)
QX9	32 Bytes	Logic output module 09 (R/W)
QX10	32 Bytes	Logic output module 10 (R/W)
QX11	32 Bytes	Logic output module 11 (R/W)
QX12	32 Bytes	Logic output module 12 (R/W)
QX13	32 Bytes	Logic output module 13 (R/W)
QX14	32 Bytes	Logic output module 14 (R/W)
QX15	32 Bytes	Logic output module 15 (R/W)
QX255	32 Bytes	Logic output CPU module (R/W)
MX0	512 Bytes	Read-only system variables (R)
MX1	512 Bytes	Read/Write system variables (R/W)
MX100	4096 Bytes	User Memory (R/W). From address 2048 to 4095 the data are retentive.

2.2 Backup memory (Retain)

SlimLine has 2048 bytes of retention memory in the **MX100** user memory and other 2000 bytes available for mnemonic variables.

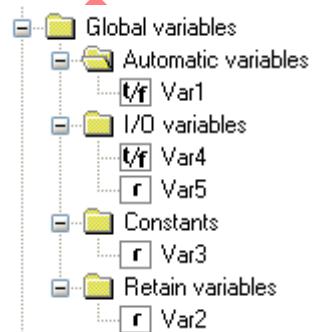
The variables allocated in the **MX100** user memory from address 2048 to 4095 are retentive, so they retain their value even during system power off.

Any mnemonic variable with the **RETAIN** attribute, will retain its value even during system power off. From the foregoing, the total area that can be allocated for the **RETAIN** variables is 2000 bytes.

	Name	Type	Address	Group	Array	Init value	Attribute	Description
1	Var1	BOOL	Auto		No	FALSE	..	Sample variable 1
2	Var2	REAL	Auto		No	0	RETAIN	Sample variable 2
3	Var3	REAL	Auto		No	12.5	CONSTANT	Sample variable 3
4	Var4	BOOL	%MD100.0		No	FALSE	..	Sample variable 4
5	Var5	REAL	%MD100.2048		No	0	..	Sample variable 5

As you can see from the picture, the **Var2** variable is declared with the **RETAIN** attribute and will maintain its value even during system power off. The **Var5** variable allocated in the user memory **MD100.2048** although does not require the **RETAIN** attribute, is retained because allocated in the retain range.

In the project window, all global variables are grouped according to their definition. One folder is for retentive variables. As you can see, only **Var2** is present. **Var5** is not present although retentive too, because it is allocated in the user memory.



2.3 Memory access

IX: Logic input process image

SlimLine reads the logic inputs at the beginning of each program loop. It is possible to access this area, by using variables of type **BOOL**. Each address represents the state of its boolean logic input. The address **IX0.0**, represents the status of the logic input 0 of extension module 0. The address **IX5.12**, represents the status of logic input 12 of extension module 5.

QX: Logic output process image

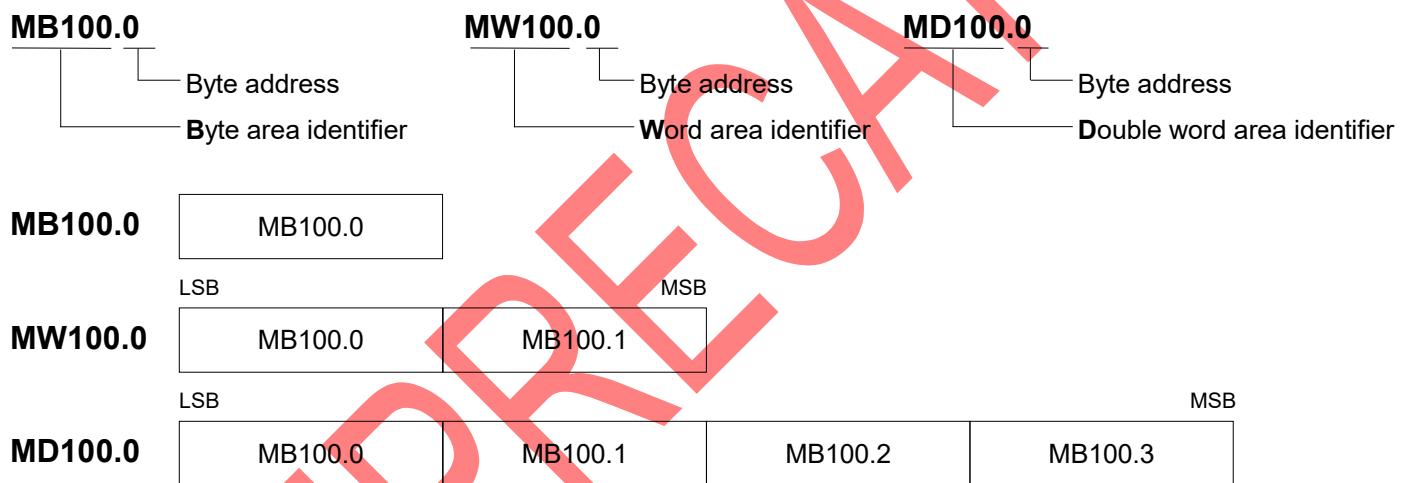
SlimLine write the logic outputs at the end of each program loop. It is possible to access this area by using variables of type **BOOL**. Each address represents the state of its logic output. The address **QX0.0**, represents the status of the logic output 0 of extension module 0. The address **QX5.12**, represents the status of the logic output 12 of extension module 5.

MX: Memory area

These areas can be accessed using all types of variables defined. Since all variables use the same memory area, it should pay attention to the size in bytes of the type defined to avoid overlapping of address.

For example, a **DWORD** variable allocated to address **MX100.10** will also use the memory space **MX100.11**, **MX100.12** and **MX100.13**. So allocating a **BYTE** variable at **MX100.11** you should occupy an area of memory already used by the previous variable.

It may be possible to allocate the address overlapping variables. For example it is possible to allocate two **BYTE** variables at the same addresses of a **WORD** variable in order to consider the MSB or LSB. Or allocate two **WORD** at the same addresses of a **DWORD** variable, in order to consider the MSW or LSW. Below a simple explanatory chart.



Warning! SlimLine is based on ARM architecture, and this type of architecture assumes that:

The 16-bit variables (**WORD**, **INT**, **UINT**) are allocated at memory addresses **divisible by 2**. So this type of variable can be allocated for example at MW100.32 but not at address MW100.33.

The 32-bit variables (**DWORD**, **DINT**, **UDINT**, **REAL**) are allocated at memory addresses **divisible by 4**. So this type of variable can be allocated for example at MD100.32 but not at address MD100.33, MD100.34, MD100.35.

This rule is automatically applied by the LogicLab compiler also regarding the **data structures**, on a definition of a data structure composed of heterogeneous variables LogicLab automatically inserts padding bytes to properly align 16 bits and 32 bits variables.

3 Data types definition

In addition to the IEC61131 standard rules, more data types have been defined, they can be used in the PLC program.

3.1 FILEP, file pointer

This data type is used by functions that execute access to the I/O system resources, a **FILEP** variable points to a resource used to reading and/or writing data. An example is a pointer to a serial port or a disk file.

3.2 SYSSERIALMODE, serial port communication mode

This data type is used by functions that perform reading and setting of serial port communication mode. The data type contains all the information to characterize the communication on serial port.

Name	Type	Description
Baudrate	UDINT	Serial port baud rate (300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200).
Parity	STRING[1]	Parity type. Possible values "E" Even, "O" Odd, "N" no parity.
DataBits	USINT	Number of data frame bits. Possible values 7, 8.
StopBits	USINT	Number of stop bits. Possible values 1, 2.
DTRManagement	USINT	DTR signal of the serial port management mode. See Serial mode definition .
DTRComplement	BOOL	FALSE: DTR normal, TRUE: DTR complemented.
EchoFlush	BOOL	FALSE: the transmitted data is returned in reception. TRUE: the transmitted data are ignored, on RS885 communications.
DTROnTime	UINT	Waiting time characters transmission on the serial port, after DTR activation (mS). This parameter has meaning only if DTRManagement is set as DTR_AUTO_W_TIMES , see Serial mode definition .
DTROffTime	UINT	Waiting time after last data transmission on the serial port and DTR deactivation (mS). This parameter has meaning only if DTRManagement is set as DTR_AUTO_W_TIMES , see Serial mode definition .

3.3 SYSCANMESSAGE, CAN message

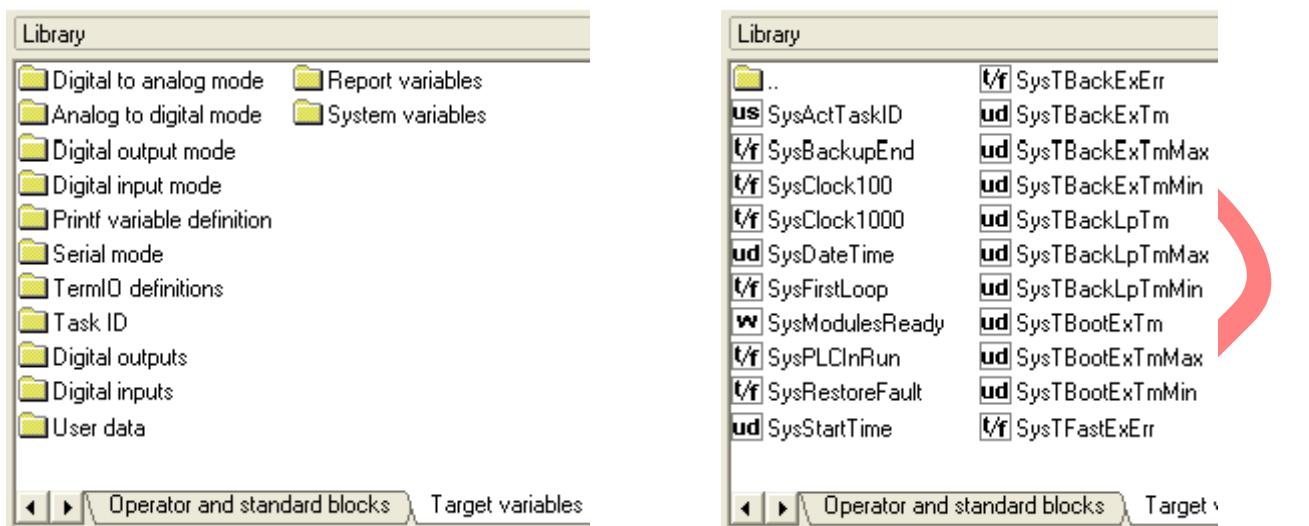
This data type is used by functions that manage the CAN controller. The structure defines the format of a CAN message.

Name	Type	Description
RmReq	BOOL	FALSE: Data frame, TRUE: Remote request.
Length	USINT	Record Length data from 0 to 8 bytes.
MsgID	UDINT	Message ID, 11 or 29 bit ID message. The 31-bit is the bit of FF.
Data	ARRAY[0..7] OF USINT	Array data message

4 System variables

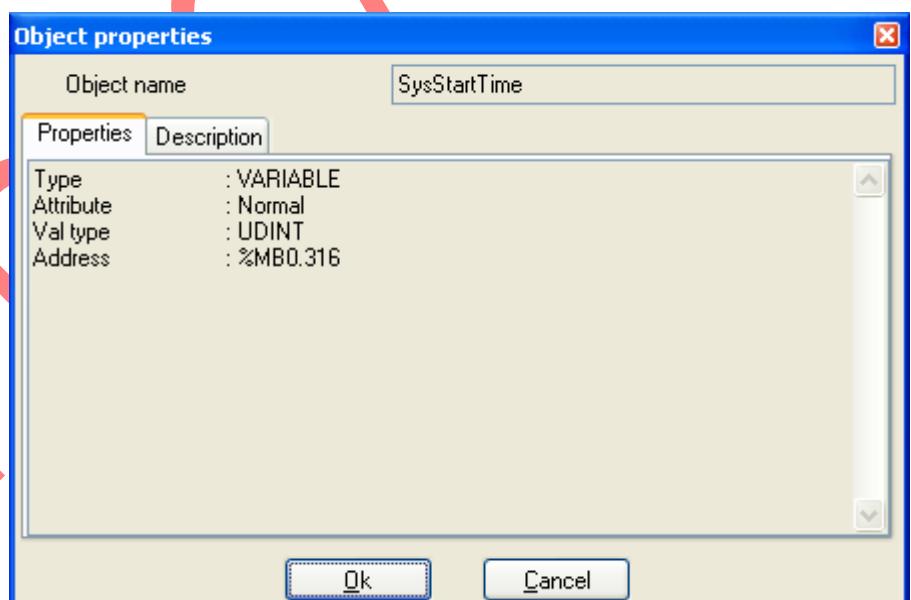
With the LogicLab development environment are published system variables that can be referenced in the program to access system information. The variables are displayed in the LogicLab library window.

If the window is not displayed, you must enable the display by the menu item **View → Tool windows → Library**. Activating the Tab **Target variables** will see a list of all published variables divided into folders. A double-click on the **System variables** folder opens the folder displaying all published variables (See picture at right).



By acting with the mouse right button on every single variable, you can view the properties dialog which shows the type and address allocation, as well as in the figure.

DEP



4.1 Read only variables (System variables)

In DB 0 are allocated read-only variables that return system information, the user program can use these variables but can not change its value.

Attention to access the variable from the user program don't use the address but always use the name as it appears in the System variables list.

Address	Name	Type	Description
%MX0.0	SysClock100	BOOL	Blinking clock with 200 mS period (change status every 100 mS).
%MX0.1	SysClock1000	BOOL	Blinking clock with 2 S period (change status every second).
%MX0.2	SysBackupEnd	BOOL	Active for a program loop at the end of a data backup cycle.
%MX0.3	SysRestoreFault	BOOL	It is activated at the power on if the backup data is in error. All data is cleared.
%MX0.4	SysPLCInRun	BOOL	Always on.
%MX0.5	SysFirstLoop	BOOL	Active for a program loop at the first execution of each PLC task.
%MX0.6	SysLLabCn	BOOL	Active if the LogicLab development environment is connected to the system.
%MX0.7	SysIsDST	BOOL	Active if in Daylight Saving Time period.
%MX0.8	SysDTSet	BOOL	Active for one background tasks execution loop on real-time clock change by operating system.
%MX0.9	SysUVSet	BOOL	Active for one background tasks execution loop on user settings change by operating system, see the example .
%MX0.10	SysAlwaysOff	BOOL	Always off variable
%MX0.11	SysAlwaysOn	BOOL	Always on variable
%MX0.12	SysFFBUspt	BOOL	Program executes at least one unsupported function or function block
%MX0.13	SysFFBPrt	BOOL	Program executes at least one protected function or function block
%MB0.64	SysActTaskID	USINT	Tasks in progress identification number, see defined types .
%MB0.65	SysErActTaskID	USINT	Task in error identification number, see defined types .
%MW0.128	SysModulesReady	UINT	Each bit of the variable, when active, indicates the presence of the module connected to the SlimLine bus.
%MW0.130	SysApIIVMajor	UINT	Application major version number
%MW0.132	SysApIIVMinor	UINT	Application minor version number
%MD0.256	SysTBackLpTm	UDINT	Current loop time of the PLC background task (uS).
%MD0.260	SysTBackLpTmMin	UDINT	Minimum loop time of the PLC background task (uS). It's possible to initialize the value by setting the SysTimelInit bit.
%MD0.264	SysTBackLpTmMax	UDINT	Maximum loop time of the PLC background task (uS). It's possible to initialize the value by setting the SysTimelInit bit.
%MD0.268	SysTBootExTm	UDINT	Current execution time of the PLC boot task (uS).
%MD0.272	SysTBootExTmMin	UDINT	Minimum loop time of the PLC boot task (uS). It's possible to initialize the value by setting the SysTimelInit bit.
%MD0.276	SysTBootExTmMax	UDINT	Maximum execution time of the PLC boot task (uS). It's possible to initialize the value by setting the SysTimelInit bit.
%MD0.280	SysTFastExTm	UDINT	Current execution time of the PLC fast task (uS).
%MD0.284	SysTFastExTmMin	UDINT	Minimum execution time of the PLC fast task (uS). It's possible to initialize the value by setting the SysTimelInit bit.
%MD0.288	SysTFastExTmMax	UDINT	Maximum execution time of the PLC fast task (uS). It's possible to initialize the value by setting the VarSysTimelInit bit.
%MD0.292	SysTSlowExTm	UDINT	Current execution time of the PLC slow task (uS).
%MD0.296	SysTSlowExTmMin	UDINT	Minimum execution time of the PLC slow task (uS). It's possible to initialize the value by setting the SysTimelInit bit.
%MD0.300	SysTSlowExTmMax	UDINT	Maximum execution time of the PLC slow task (uS). It's possible to initialize the value by setting the SysTimelInit bit.
%MD0.304	SysTBackExTm	UDINT	Current execution time of the PLC background task (uS).
%MD0.308	SysTBackExTmMin	UDINT	Minimum execution time of the PLC background task (uS). It's possible to initialize the value by setting the SysTimelInit bit.

Address	Name	Type	Description
			initialize the value by setting the SysTimeInit bit.
%MD0.312	SysTBackExTmMax	UDINT	Maximum execution time of the PLC background task (uS). It's possible to initialize the value by setting the SysTimeInit bit.
%MD0.316	SysStartTime	UDINT	Date and time to start execution of the PLC program (Epoch time).
%MD0.320	SysTime	UDINT	System time, it increments each 1 mS, reached the maximum the value is reinitialized.
%MD0.324	SysTFastLpTm	UDINT	PLC fast task loop time. The time is set by using the SysSetTaskLpTime function.
%MD0.328	SysTSlowLpTm	UDINT	PLC slow task loop time. The time is set by using the SysSetTaskLpTime function.
%MD0.332	SysApplID	UDINT	Application ID, is a unique number that identifies the user program currently running on your system.
%MD0.336	SysMfcCode	UDINT	Manufacturer code, this code must be required with the product. If undefined code 0 is returned.
%MD0.340	SysCustomerCode	UDINT	Customer code, this code can be set by the user who has administrative access to the system. If undefined code 0 is returned.
%MD0.344	SysErCode	UDINT	Execution program error code.
%MD0.348	SysErTime	UDINT	Date and time when the error occurred (Epoch time).
%MD0.352	SysSerialNr	UDINT	Product serial number.
%MD0.356	SysApplIBTime	UDINT	Build time (Epoch time).
%MD0.360	SysUniqueId	UDINT	Product unique ID, see note .
%MB0.512	SysCode	STRING[10]	Product code.
%MB0.523	SysFVersion	STRING[10]	Product firmware version.
%MB0.534	SysErInfos	STRING[32]	Additional information about the error.
%MB0.567	SysApplName	STRING[10]	Application name.
%MB0.578	SysUSetA	STRING[16]	Value set by user in Set(A) variable, see the example .
%MB0.595	SysUSetB	STRING[16]	Value set by user in Set(B) variable, see the example .
%MB0.612	SysUSetC	STRING[16]	Value set by user in Set(C) variable, see the example .
%MB0.629	SysUSetD	STRING[16]	Value set by user in Set(D) variable, see the example .

4.2 Read and write variables (System variables)

In DB 1 are allocated read-only variables that allow to change the system operation, the user program can use these variables.

Attention to access the variable from the user program don't use the address but always use the name as it appears in the System variables list.

Address	Name	Type	Description
%MX1.0	SysTimeInit	BOOL	Activated by the user program or by debug allows to initialize the loop and execution times of PLC tasks. The variable is automatically reset by the system.
%MD1.256	SysDateTime	UDINT	System local date and time (Epoch time), according to the defined time zone and daylight. By changing the value in the user program, will be automatically updated the real time clock.
%MD1.260	SysLastError	UDINT	Last error, returns the value of the last error that occurred in the execution of a function or function block, Object IDs .
%MB1.512	SysUInfoA	STRING[16]	Value returned to user in Set(A) variable, see the example .
%MB1.529	SysUInfoB	STRING[16]	Value returned to user in Set(B) variable, see the example .
%MB1.546	SysUInfoC	STRING[16]	Value returned to user in Set(C) variable, see the example .
%MB1.563	SysUInfoD	STRING[16]	Value returned to user in Set(D) variable, see the example .

DEPRECATE

4.3 Product unique ID

Each product has a unique ID that is returned in the ***UniqueId*** variable, the value is based on the type of product and its serial number. The formula for the UniqueID calculation is the following:

$$\text{UniqueId} = (131072 * \text{PType}) + \text{Serial number}$$

For example a MPS050A030 with serial number 125 has UniqueID=1310845. Here the **Ptype** table.

PType	Code	Description
0000	MPS046A000	SlimLine (Lite version)
0001	MPS046A100	SlimLine (Rs485 version)
0002	MPS046A200	SlimLine (CAN version)
0003	MPS048A100	SlimLine ARM9 RS485 (Linux Open)
0004	MPS048A200	SlimLine ARM9 CAN (Linux Open)
0005	MPS049A100	SlimLine ARM9 RS485 (Linux PLC)
0006	MPS049A200	SlimLine ARM9 CAN (Linux PLC)
0007	MPS050A000	SlimLine Low Cost ARM7 (Vers. Lite)
0008	MPS050A010	SlimLine Low Cost ARM7 (Vers. Base)
0009	MPS050A020	SlimLine Low Cost ARM7 (Vers. Full RS485)
0010	MPS050A030	SlimLine Low Cost ARM7 (Vers. Full CAN)
0011	PCB123B000	SlimLine OEM (Lite version)
0012	PCB123B100	SlimLine OEM (Rs485 version)
0013	PCB123B200	SlimLine OEM (CAN version)
0014	MPS046B000	SlimLine (Lite version)
0015	MPS046B100	SlimLine (Rs485 version)
0016	MPS046B200	SlimLine (CAN version)
0017	PCB123D000	SlimLine OEM (Lite version)
0018	PCB123D100	SlimLine OEM (Rs485 version)
0019	PCB123D200	SlimLine OEM (CAN version)
0020	PCB131A000	SlimLine ARM7 Compact Relay CPU Board (Lite vers.)
0021	PCB131A010	SlimLine ARM7 Compact Relay CPU Board (Base vers.)
0022	PCB131A020	SlimLine ARM7 Compact Relay CPU Board (Full RS485 v.)
0023	PCB131A030	SlimLine ARM7 Compact Relay CPU Board (Full CAN v.)
0024	MPS051A000	Netlog III Base Relay
0025	MPS051A001	Netlog III Full Relay (Rs485 version)
0026	MPS051A011	Netlog III Full Relay and Display (Rs485 version)
0027	MPS051A002	Netlog III Full Relay (CAN version)
0028	MPS051A012	Netlog III Full Relay and Display (CAN version)
0029	MPS051A300	Netlog III Base Static
0030	MPS051A301	Netlog III Full Static (Rs485 version)
0031	MPS051A311	Netlog III Full Static and Display (Rs485 version)
0032	MPS051A302	Netlog III Full Static (CAN version)
0033	MPS051A312	Netlog III Full Static and Display (CAN version)

5 Data definition

In addition to the folders of the system variables, there are also folders with identifiers of type information needed to identify a given system data.

5.1 Variable types definition

Each variable type is defined with a value that identifies it, the value is indicated by definitions that can be found in the the **Variable types definition** folder.

Define	Type	Value	Description
BOOL_TYPE	USINT	10	Boolean variable (BOOL), 1 bit can have only one meaning FALSE or TRUE.
BYTE_TYPE	USINT	20	Byte variable (BYTE) 8 bits unsigned, range from 0 to 255.
SINT_TYPE	USINT	21	Byte variable (SINT) 8-bit signed, range from -128 to +127.
USINT_TYPE	USINT	22	Byte variable (USINT) 8 bits unsigned, range from 0 to 255.
WORD_TYPE	USINT	30	Word variable (WORD) 16 bits unsigned, range from 0 to 65535.
INT_TYPE	USINT	31	Word variable (INT) 16 bits signed, range from -32768 to 32767
UINT_TYPE	USINT	32	Word variable (UINT) 16 bits unsigned, range from 0 to 65535.
DWORD_TYPE	USINT	40	Double word variable (DWORD) 32 bits unsigned, range from 0 to 4294967295.
DINT_TYPE	USINT	41	Double word variable (DINT) 32 bits signed, range from -2147483648 to 2147483647.
UDINT_TYPE	USINT	42	Double word variable (UDINT) 32 bits unsigned, range from 0 to 4294967295.
REAL_TYPE	USINT	43	Floating variable (REAL) 32 bits signed, range from -3.40E+38 to +3.40E+38.
STRING_TYPE	USINT	50	String variable (STRING).

5.2 Task ID definition

The PLC tasks are identified by a value, the value is indicated by definitions that can be found in the the **Task ID definition** folder.

Define	Type	Value	Description
ID_TASK_BOOT	USINT	0	Identifies the task Boot PLC. This task runs only on the first loop of user program.
ID_TASK_BACK	USINT	1	Identifies the background task. This task runs in the background to the slow and fast task. The loop time of this task is not fixed but depends on the workload of the CPU executing other tasks.
ID_TASK_SLOW	USINT	2	Identifies the slow task. The loop time can be set with the SysSetTaskLpTime function. The default time is 10 mS.
ID_TASK_FAST	USINT	3	Identifies the fast task. The loop time can be set with the SysSetTaskLpTime function. The default time is 1 mS.

5.3 TermIO, definitions for terminal I/O

In the management of terminal I/O are used definitions that can be found in the folder ***TermIO definition***.

Define	Type	Value	Description
NULL	FILEP	0	Identifies a pointer to void. Used as a return of some functions in case of error.
EOF	INT	-1	Identifies the end of file. Used as a return value of some functions in case of end of file reached.

5.4 FSeek origin definition

In the file seek are used definitions that can be found in the ***FSeek origin definition*** folder.

Define	Type	Value	Description
ID_SEEK_SET	USINT	0	Inizio del file
ID_SEEK_CUR	USINT	1	Posizione corrente file
ID_SEEK_END	USINT	2	Fine del file

5.5 Serial mode definition

In the management of terminal I/O make good use of definitions that can be found in the ***Serial mode definition*** folder.

Define	Type	Value	Description
DTR_OFF	USINT	0	Member value DTRManagement of SYSSERIALMODE : it sets DTR always off.
DTR_ON	USINT	1	Member value DTRManagement of SYSSERIALMODE : it sets DTR always on.
DTR_AUTO_WO_TIMES	USINT	2	Member value DTRManagement of SYSSERIALMODE : it sets automatic DTR without time.
DTR_AUTO_W_TIMES	USINT	3	Member value DTRManagement of SYSSERIALMODE : it sets automatic DTR with time.

5.6 CAN bit rate definition

To set the bit rates on the CAN controller a definitions can be found in the ***CAN bit rate definition*** folder.

Define	Type	Value	Description
CAN_50KBIT	USINT	0	Bit rate 50 KBit
CAN_100KBIT	USINT	1	Bit rate 100 KBit
CAN_125KBIT	USINT	2	Bit rate 125 KBit
CAN_250KBIT	USINT	3	Bit rate 250 KBit
CAN_500KBIT	USINT	4	Bit rate 500 KBit
CAN_1MBIT	USINT	5	Bit rate 1 MBit

5.7 Digital input mode

To set the acquisition mode of digital input modules a definitions can be found in the **Digital input mode** folder.

Define	Type	Value	Description
DI_8_LL	USINT	1	Read 0-7 input mode (Debounced)
DI_8_L	USINT	2	Read 8-15 input mode (Debounced)
DI_8_M	USINT	3	Read 16-23 input mode (Debounced)
DI_8_MM	USINT	4	Read 24-31 input mode (Debounced)
DI_16_L	USINT	5	Read 0-15 input mode (Debounced)
DI_16_M	USINT	6	Read 16-31 input mode (Debounced)
DI_32	USINT	7	Read 0-31 input mode (Debounced)
DI_I_8_LL	USINT	11	Read 0-7 input mode (Immediate)
DI_I_8_L	USINT	12	Read 8-15 input mode (Immediate)
DI_I_8_M	USINT	13	Read 16-23 input mode (Immediate)
DI_I_8_MM	USINT	14	Read 24-31 input mode (Immediate)
DI_I_16_L	USINT	15	Read 0-15 input mode (Immediate)
DI_I_16_M	USINT	16	Read 16-31 input mode (Immediate)
DI_I_32	USINT	17	Read 0-31 input mode (Immediate)

5.8 Digital output mode

To set the management mode of digital outputs modules a definitions can be found in the **Digital output mode** folder.

Define	Type	Value	Description
DO_8_LL	USINT	1	Write 0-7 output mode
DO_8_L	USINT	2	Write 8-15 output mode
DO_8_M	USINT	3	Write 16-23 output mode
DO_8_MM	USINT	4	Write 24-31 output mode
DO_16_L	USINT	5	Write 0-15 output mode
DO_16_M	USINT	6	Write 16-31 output mode
DO_32	USINT	7	Write 0-31 output mode

5.9 Analog to digital mode

To set the acquisition mode of analog inputs modules a definitions can be found in the **Analog to digital mode** folder.

Define	Type	Value	PCB099 PCB122	PCB126	Description
AD_IDLE	USINT	0	•	•	Idle mode
AD_CURR_0_20_COMMON	USINT	3	•	•	Current from 0 to 20 mA (Common mode)
AD_CURR_0_20_DIFFER	USINT	6	•	•	Current from 0 to 20 mA (Differential mode)
AD_CURR_4_20_COMMON	USINT	4	•	•	Current from 4 to 20 mA (Common mode)
AD_CURR_4_20_DIFFER	USINT	13	•	•	Current from 4 to 20 mA (Differential mode)
AD_NI1000_DIFFER	USINT	12		•	Ni1000 sensor Celsius degree (Differential mode)
AD_NI1000_DIN_43760	USINT	48		•	Ni1000 DIN_43760 standard Celsius degree
AD_PT100_AMERICAN	USINT	33		•	Pt100 American standard Celsius degree
AD_PT100_DIFFER	USINT	10	•	•	Pt100 sensor Celsius degree (Differential mode)
AD_PT100_DIN_43760	USINT	32	•	•	Pt100 DIN_43760 standard Celsius degree
AD_PT100_IEC_60751	USINT	35		•	Pt100 IEC-60751 standard Celsius degree
AD_PT100_ITS_90	USINT	34		•	Pt100 ITS-90 standard Celsius degree
AD_PT1000_AMERICAN	USINT	41		•	Pt1000 American standard Celsius degree
AD_PT1000_DIFFER	USINT	11	•	•	Pt1000 sensor Celsius degree (Differential mode)
AD_PT1000_DIN_43760	USINT	40	•	•	Pt1000 DIN_43760 standard Celsius degree
AD_PT1000_IEC_60751	USINT	43		•	Pt1000 IEC-60751 standard Celsius degree
AD_PT1000_ITS_90	USINT	42		•	Pt1000 ITS-90 standard Celsius degree
AD_RESISTOR_300_OHM	USINT	110	•	•	Resistor up to 300 Ohm
AD_RESISTOR_5000_OHM	USINT	111		•	Resistor up to 5000 Ohm
AD_THERMOCOUPLE_B	USINT	64		•	Thermocouple B type Celsius degree
AD_THERMOCOUPLE_E	USINT	65		•	Thermocouple E type Celsius degree
AD_THERMOCOUPLE_J	USINT	66		•	Thermocouple J type Celsius degree
AD_THERMOCOUPLE_K	USINT	67		•	Thermocouple K type Celsius degree
AD_THERMOCOUPLE_N	USINT	68		•	Thermocouple N type Celsius degree
AD_THERMOCOUPLE_R	USINT	69		•	Thermocouple R type Celsius degree
AD_THERMOCOUPLE_S	USINT	70		•	Thermocouple S type Celsius degree
AD_THERMOCOUPLE_T	USINT	71		•	Thermocouple T type Celsius degree
AD_VIN_VREF_G_1	USINT	90		•	Voltage in/Voltage reference with gain 1
AD_VIN_VREF_G_128	USINT	97		•	Voltage in/Voltage reference with gain 128
AD_VIN_VREF_G_16	USINT	94		•	Voltage in/Voltage reference with gain 16
AD_VIN_VREF_G_2	USINT	91		•	Voltage in/Voltage reference with gain 2
AD_VIN_VREF_G_32	USINT	95		•	Voltage in/Voltage reference with gain 32
AD_VIN_VREF_G_4	USINT	92		•	Voltage in/Voltage reference with gain 4
AD_VIN_VREF_G_64	USINT	96		•	Voltage in/Voltage reference with gain 64
AD_VIN_VREF_G_8	USINT	93		•	Voltage in/Voltage reference with gain 8
AD_VOLT_0_1_COMMON	USINT	5	•	•	Voltage from 0 to 1 V (Common mode)
AD_VOLT_0_1_DIFFER	USINT	7	•	•	Voltage from 0 to 1 V (Differential mode)
AD_VOLT_0_10_COMMON	USINT	2	•	•	Voltage from 0 to 10 V (Common mode)
AD_VOLT_0_10_DIFFER	USINT	9	•	•	Voltage from 0 to 10 V (Differential mode)
AD_VOLT_0_125_COMMON	USINT	1	•	•	Voltage from 0 to 1.25 V (Common mode)
AD_VOLT_0_125_DIFFER	USINT	8	•	•	Voltage from 0 to 1.25 V (Differential mode)

5.10 Digital to analog mode

To set the output mode of analog output modules a definitions can be found in the **Digital to analog mode** folder.

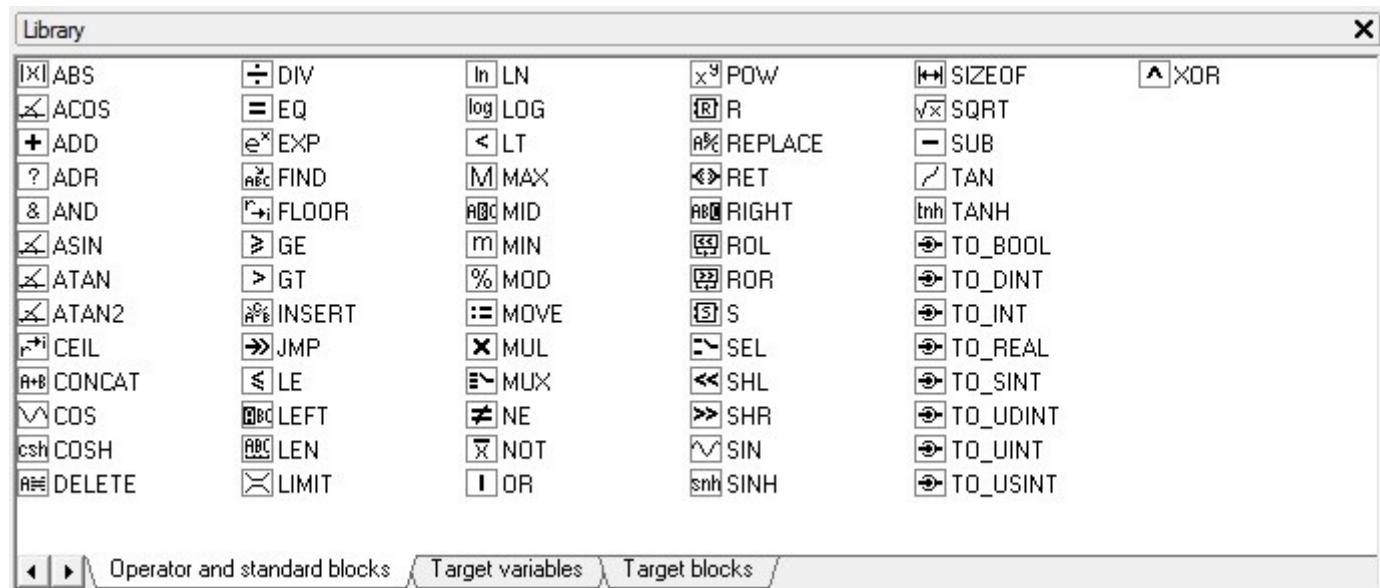
Define	Type	Value	PCB099 PCB122	PCB126	Description
DA_CURR_0_20	USINT	5		●	Current from 0 to 20 mA
DA_CURR_4_20	USINT	6		●	Current from 4 to 20 mA
DA_VOLT_0_10	USINT	1	●	●	Voltage from 0 to 10 V
DA_VOLT_0_5	USINT	2		●	Voltage from 0 to 5 V
DA_VOLT_M10_10	USINT	3		●	Voltage from -10 to +10 V
DA_VOLT_M5_5	USINT	4		●	Voltage from -5 to +5 V

5.11 Spy mode, spy mode definition

The definitions of spying data so you can find the folder ***Spy mode definition***.

6 Functions defined by LogicLab

LogicLab natively supports a number of functions that can be used by programs written in the different languages that are supported. To use these functions just pick the desired function from the **Operator and standard blocks** draw. In textual languages such as IL and ST is also possible to write the function exactly as it is defined.

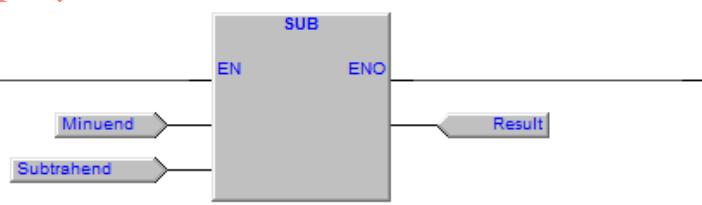


Here are some examples of use in various languages.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Minuend	UINT	Auto	No	0	..	Minuend
2	Subtrahend	UINT	Auto	No	0	..	Subtrahend
3	Result	UINT	Auto	No	0	..	Result

LD example



IL example

```
LD Minuend
SUB Subtrahend
ST Result
```

ST example

```
Result:=SUB(Minuend, Subtrahend); (* Subtraction using SUBB *)
Result:=Minuend-Subtrahend; (* Subtraction as before *)
```

6.1 Mathematical and trigonometric functions

LogicLab supports all the mathematical functions required by the standard IEC 61131-3 Part 3: Programming Languages. Below a list of supported functions and an example of use in ST language.

ABS	ANY_NUM ABS(ANY_NUM) It calculates the absolute value of a number.
	Result:=ABS(-10.5); (* Result is 10.5 *)
SQRT	ANY_REAL SQRT(ANY_REAL) It calculates the square root of a number.
	Result:=SQRT(9.0); (* Result is 3.0 *)
LN	ANY_REAL LN(ANY_REAL) It calculates the natural logarithm (Base "e" 2.71828) of a number.
	Result:=LN(10.0); (* Result is 2.30259 *)
LOG	ANY_REAL LOG(ANY_REAL) It calculates the logarithm (base "10") of a number.
	Result:=LOG(10.0); (* Result is 1.0 *)
EXP	ANY_REAL EXP(ANY_REAL) It calculates number raised to "e" 2,71828.
	Result:=EXP(1.0); (* Result is 2.71828 *)
SIN	ANY_REAL SIN(ANY_REAL) It calculates the sine of an angle in radians.
	Result:=SIN(1.57); (* Angle is 90°, Result is 1.0 *)
COS	ANY_REAL COS(ANY_REAL) It calculates the cosine of an angle in radians.
	Result:=COS(3.1416); (* Angle is 180°, Result is -1.0 *)
TAN	ANY_REAL TAN(ANY_REAL) It calculates the tangent of an angle in radians.
	Result:=TAN(0.7854); (* Angle is 45°, Result is 1.0 *)
ASIN	ANY_REAL ASIN(ANY_REAL) It calculates the arc sine of an angle in radians.
	Result:=ASIN(1.0); (* Result is: 1.5708 *)
ACOS	ANY_REAL ACOS(ANY_REAL) It calculates the arc cosine of an angle in radians.
	Result:=ACOS(-1.0); (* Result is 3.14159 *)
ATAN	ANY_REAL ATAN(ANY_REAL) It calculates the arc tangent of an angle in radians.
	Result:=ATAN(1.0); (* Result is 3.14159 *)
ADD	ANY_NUM ADD(ANY_NUM, ANY_NUM) It executes the sum of two numbers.
	Result:=ADD(1.0, 2.0); (* Result is 3.0 *)
MUL	ANY_NUM MUL(ANY_NUM, ANY_NUM) It executes the multiplication of two numbers.
	Result:=MUL(1.0, 2.0); (* Result is 2.0 *)
SUB	ANY_NUM SUB(ANY_NUM, ANY_NUM) It executes the subtraction of two numbers.
	Result:=SUB(2.0, 1.0); (* Result is 1.0 *)

DIV**ANY_NUM DIV(ANY_NUM, ANY_NUM)**

It executes the division of two numbers.

```
Result:=DIV(2.0, 1.0); (* Result is 2.0 *)
```

About the modulus operator MOD please note that the modular arithmetic is applied only to integers, where numbers "wrap themselves about themselves" every time they reach the multiples of a given number n, said module.

The operation Result:=x MOD 10, will returns values between 0 and 10 to any value of x.

The operation Result:=x MOD 1000, will returns values between 0 and 1000 to any value of x.

DEPRECATE

6.2 String functions

LogicLab supports all the string functions required by the standard IEC 61131-3 Part 3: Programming Languages. Below a list of supported functions and an example of use in ST language.

CONCAT

STRING CONCAT(STRING S0, STRING S1)

Concatenate strings S0 and S1

```
AString:='Hello ';
BString:='World !';
CString:=CONCAT(AString, BString); (* CString is 'Hello World !' *)
CString:=CONCAT(AString, 'World !'); (* CString is 'Hello World !' *)
```

DELETE

STRING DELETE(STRING IN, ANY_INT L, ANY_INT P)

Delete from the string IN L characters starting from position P in backward.

```
AString:='Hello World !';
BString:=DELETE(AString, 2, 3); (* BString is 'Heo World !' *)
```

FIND

UINT FIND(STRING S0, STRING S1)

Search the location of the first occurrence of S1 in S0. If not found returns 0.

```
AString:='Hello World !';
i:=FIND(AString, 'World'); (* Result is: 7 *)
j:=FIND('Hello World World !', 'World'); (* Result is 7 *)
k:=FIND('World', 'Hello'); (* Result is: 0 *)
```

INSERT

STRING INSERT(STRING S0, STRING S1, ANY_INT P)

Inserts in the string S1 the string S0 string starting from position P.

```
AString:='Hello everybody';
BString:='World !';
CString:=INSERT(AString, BString, 6); (* CString is 'Hello World !'*)
```

LEFT

STRING LEFT(STRING IN, ANY_INT L)

It returns the L leftmost characters of the string IN.

```
AString:='Hello World !';
BString:=LEFT(AString, 7); (* BString is 'Hello W'
```

LEN

UINT LEN(STRING IN)

It returns the length (Number of characters) of string IN.

```
AString:='Hello World !';
i:=LEN(AString); (* i is: 13 *)
```

MID

STRING MID(STRING IN, ANY_INT L, ANY_INT P)

It returns the L characters of the string IN, starting at character position P.

```
AString:='Hello World !';
BString:=MID(AString, 5, 7); (* BString is 'World' *)
```

REPLACE

STRING REPLACE(STRING S0, STRING S1, ANY_INT L, ANY_INT P)

Replaces L characters of the string S0 with the string S1, starting from the position P.

```
AString:='Hello World !';
BString:=REPLACE(AString, 'to you ', 5, 7);
```

RIGHT

STRING RIGHT(STRING IN, ANY_INT L)

It returns the L rightmost characters of the string IN.

```
AString:='Hello World !';
BString:=RIGHT(AString, 7); (* BString is 'World !' *)
```

7 Functions and FBs

Functions

The functions have a variable number of input data and one only data output. To use them, just put them in the LD and FBD programs and connect them to the variables. In the IL programs, they must be called with the statement CAL. In the ST programs simply indicate the name to be executed.

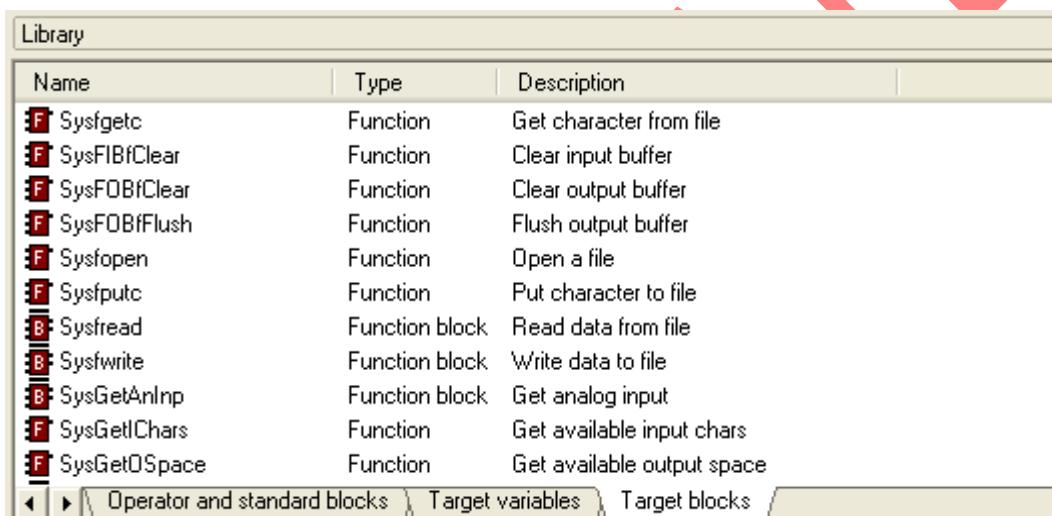
Function Blocks

The FBs unlike functions, allocate a variable in the program that contains all the variables of input and output handled by the function block. To use them just put them in the LD and FBD programs and connect them to the variables. In the IL programs they must be called with the statement CAL. In the ST programs simply indicate the name to be executed.

7.0.1 Functions and embedded FBs

With the LogicLab development environment, functions and function blocks (FB) that allow to access to the embedded hardware and software of **SlimLine** system, are provided . The functions and FBs are shown in the window LogicLab libraries.

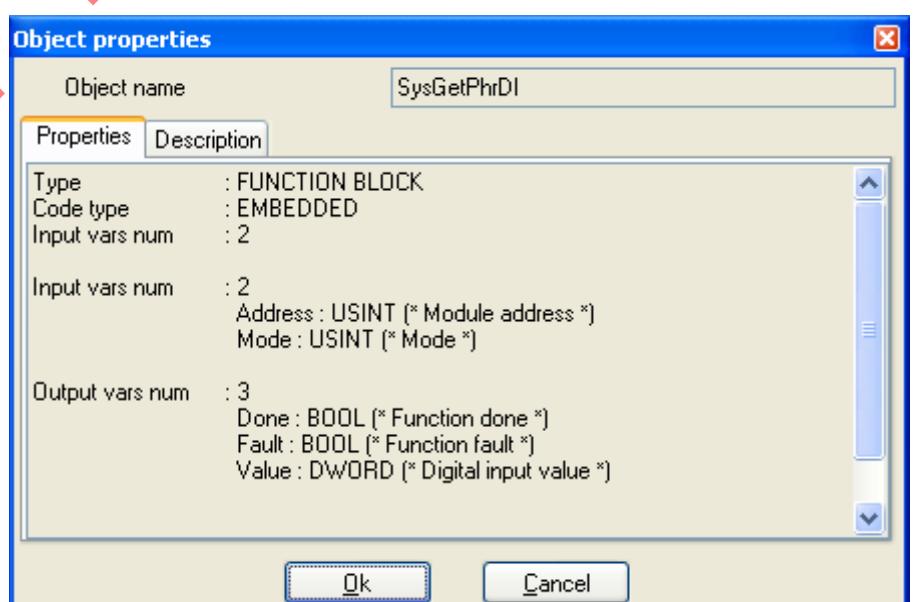
If the window is not displayed, you must enable the display of it from the menu item **View → Tool windows → Library**. Activating the **Target blocks** Tab, will be visible a list of all functions (marked as **F**) and function blocks (marked as **B**) embedded.



Name	Type	Description
F Sysgetc	Function	Get character from file
F SysFIBfClear	Function	Clear input buffer
F SysFOBfClear	Function	Clear output buffer
F SysFOBfFlush	Function	Flush output buffer
F Sysopen	Function	Open a file
F Sysputc	Function	Put character to file
B Sysread	Function block	Read data from file
B Sysfwrite	Function block	Write data to file
B SysGetAnInp	Function block	Get analog input
F SysGetChars	Function	Get available input chars
F SysGetOSpace	Function	Get available output space

Operator and standard blocks Target variables Target blocks

Acting with the right mouse button on any single function or function block, you can view the properties window that lists the input variables and the return of function, while for function blocks the the input and output variables are shown, as well as in the figure.



7.0.2 Libraries

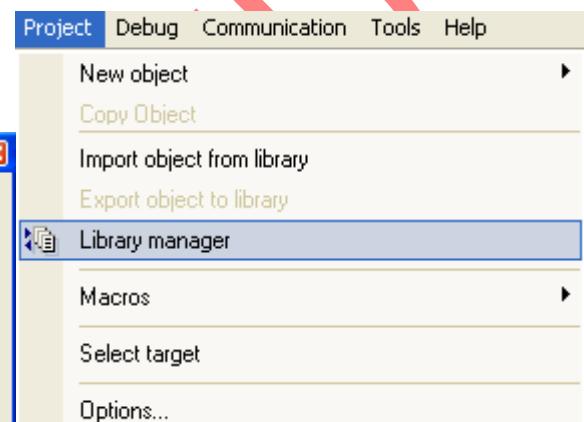
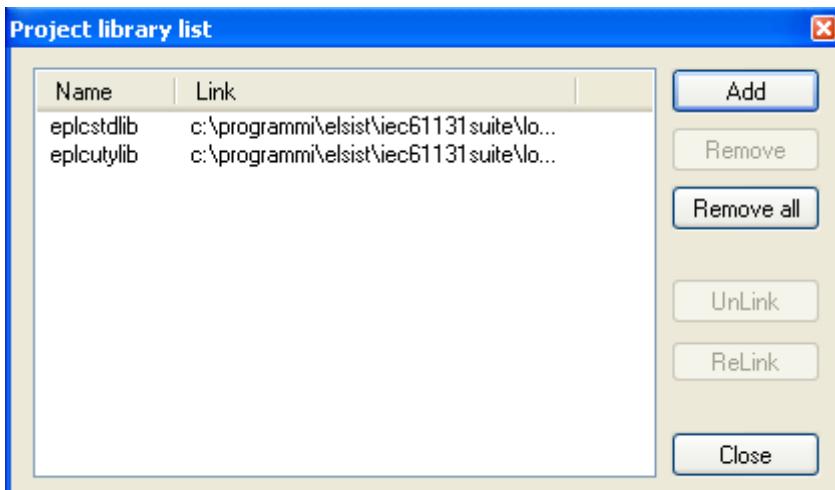
In addition to the functions and embedded FBs, some libraries are provided. They contain a variety of functions and function blocks that can be used in your program. The libraries are provided with LogicLab in the installation directory **ProgramFiles\Elsist\IEC61131Suite\LogicLab2p0\Libraries**, but you can also use libraries provided subsequently or which have been downloaded from the site. There are two ways to use the libraries:

Import library: In this way, all the items in the library are imported into your program and the objects can thus be used in the program. This is a good solution. The drawback is to increase the size of the LogicLab project file (*.ppjs), as well as its own program must contain all the objects of the imported library. **The executable generated will contain only the used objects, however.**

Import objects: In this way you can import only the objects from a library (functions, FBs, etc.) that affect, which will become an integral part of their project.

7.0.3 Import library

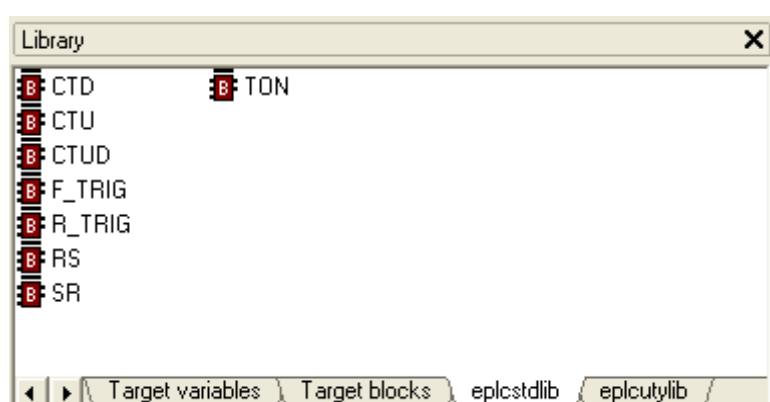
In this mode, all objects are imported into the library. To import the entire library in your program, select the menu item **Project** → **Library manager** and will open a window like the one below.



Acting on the **Add** button, opens a browser window of the disk. Choose the directory where is placed the library, and select the file to be imported.

Acting on the **Close** button, in the LogicLab **Library** window (**Ctrl-L**) will show an additional tab, one for each imported library.

Simply drag the desired object in your project to use it.



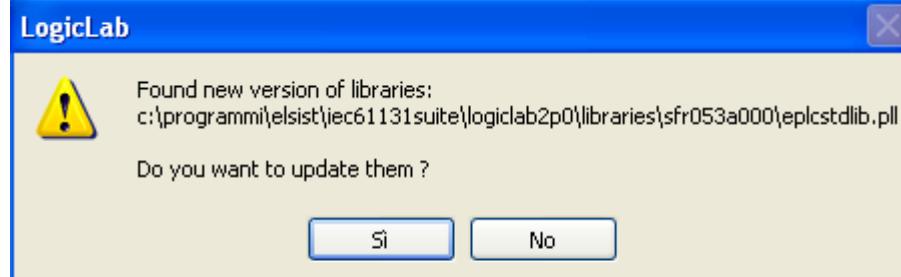
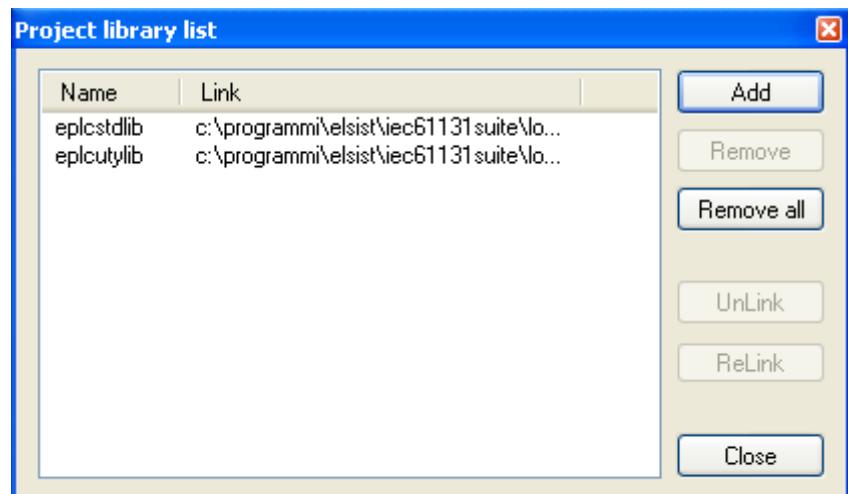
7.0.4 Import a library

Running the import library in your project as described in the previous chapter, all objects in the imported library are transferred into your project file (*.ppjs), but is still maintained a link to the source library as seen from the window to the side.

When the library is modified with a newer version, this allows to make the automatic update of the new library in your project.

If the source library is no longer present or has been moved from the position where it was imported, LogicLab will no longer control without reporting errors.

Through the **Project → Library manager** menu that opens window to the side, as shown, you can select the various libraries and the **UnLink** button to remove the link or with the **ReLink** button, to perform a new location where the library is placed.



Opening the project, LogicLab controls all imported libraries and if one or more sources are more recent versions than imported, it displays a warning message asking whether to confirm or not the updated libraries..

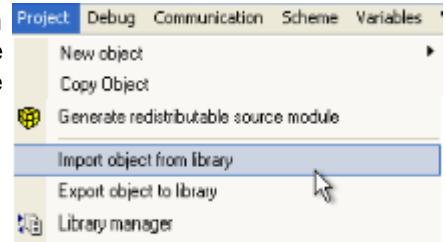
If you confirm to update, all the objects of the imported library present in the project, will be overwritten with the objects in the newer library and any new

objects are automatically imported.

DEPREC

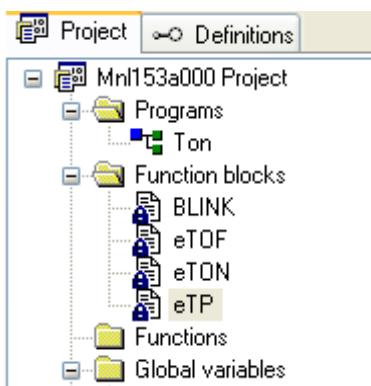
7.0.5 Import library objects

To import objects from libraries, in your program you must select the menu item **Project → Import object from library**. This will open a browser window of the disk. Choose the directory where the library is placed, and select the library file from which you want to import objects.

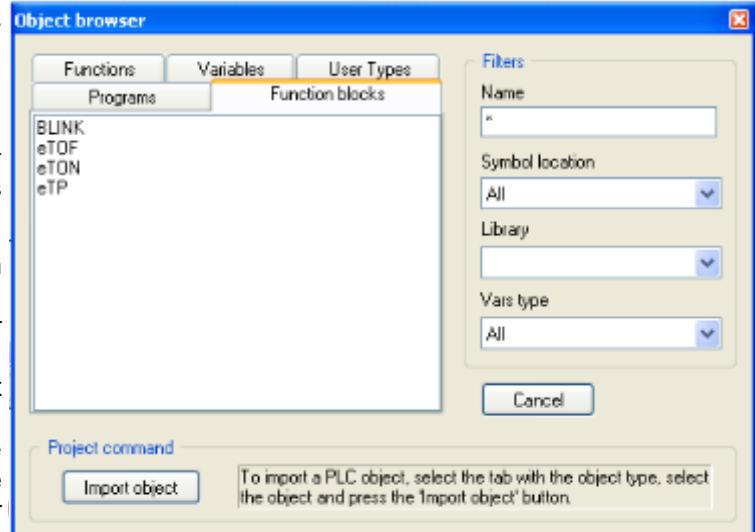


This will open the **Object browser** window that allows you to view all the objects in the library. Selecting the different tabs you can view all these items in the library sorted by name.

A click of the mouse highlights the desired object or objects. With the **Import object** button, selected objects will be included in the program.



As you can see from the photo on the right, some objects appear with a padlock symbol. This means that objects are protected, they can not be modified. Once imported into your program, the objects will be included in the program and you can use them on any PC even if you do not have the original library.



DEPRECATE

7.0.6 Considerations about links to library and import of objects

As seen in previous sections, to use functions and/or function blocks from library, you can use two different methods: import only the desired object or the whole library in your project.

In both cases the object will be included in your project. In this way you are sure that in the future with newer versions of the library, will always be able to recompile the project using the object with which it was developed and tested.

If you are replacing the object with a newer version of the same, we will use a different approach depending on whether the object is present in a linked library or has been imported.

Linked library

As indicated earlier, linked libraries maintain a reference to the source library. In the installation path of LogicLab, the libraries are in directories whose name represents the version. In this way newer versions of the library can be distributed, but the project to its reopening will always check with the original version without an automatic upgrade.

To update a linked object in a library, a **Relink** the new version of the library must be perform. **Warning!** This will update all objects in the library.

Imported object

In the case of imported object, to update, simply remove the object from the current project and run an import of the same object from the new version of the library.

Conclusions

In general you should not connect the library. Instead, import the individual objects into your project. This allows an easier update management.

Some libraries contain a set of objects (functions and function blocks) that are widely used, in this case it is always advisable to link these libraries. Here is the list of libraries that you may want to connect to the project:

Libreria	Code	Description
ePLCStdLib	SFR053*000	IEC61131 standard library. Contains functions and function blocks defined by IEC61131 and not present in the embedded library of the product.
ePLCAuxLib	SFR058*000	Auxiliary library. Contains functions and function blocks of varying usefulness.

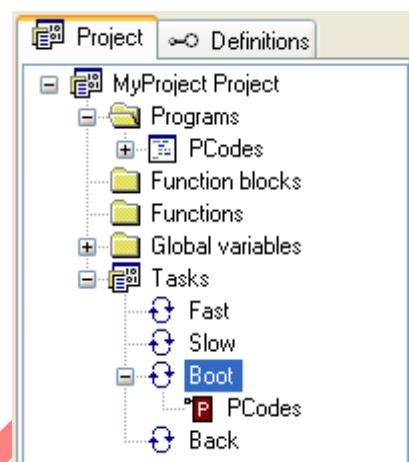
7.0.7 Functions and function blocks protection

Some library functions and/or function blocks can be protected by a code which must be ordered separately. To enable the execution you must unlock them defining a code (alphabetic string of 18 characters) with the **SysPCodeAccept** function.

The function should be performed only once passing the security code. If the code is correct, the function returns **TRUE** and its function will be unprotected until the next restart of the program. It is possible to make multiple calls to the function, one for each security code.

The advice is to place multiple calls to the function in a program that will run in the boot task thus before each call to other programs, ensuring the unlock of the desired functions.

On the side you can see how the program where are the **PCodes** security codes definition, is associated with the boot task. Here's the source code of the program **PCodes** realized in ST. Of course, the same codes are fictional so if you run the function, the **SysPCodeAccept** function will always return **FALSE**.



Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	CodesOk	BOOL	Auto	[0..2]	3(0)	..	Protection codes ok

ST example

```
(* Check the protection codes. *)
CodesOk[0]:=SysPCodeAccept('abcdefghijklmnpqr'); (* Protection code ok (Function 1) *)
CodesOk[1]:=SysPCodeAccept('rqponmlkjihgfedcba'); (* Protection code ok (Function 2) *)
CodesOk[2]:=SysPCodeAccept('abcdefghijklmnpqr'); (* Protection code ok (Function 3) *)

(* [End of file] *)
```

Normally the protected functions and function blocks can operate in demo mode for a certain period of time on their first run after the system power on. After the test period, the operation ends, and they generates an error that is detectable by the function **SysGetLastError**.

7.1 Functions and FBs for Flip/Flop management

7.1.1 F_TRIG, Falling edge trigger

Type	Library
FB	ePLCStdLib_B000

This function block enables the **Q** output for a program loop on the falling edge of the **CLK** clock.



CLK (BOOL) Clock. On the falling edge signal, the Q output is turned on for a program loop.

Q (BOOL) Output. Active for a program loop on the falling edge of the CLK clock input.

Examples

On the falling edge of the digital **Di00M00**, digital output **Do00M00** became enabled for a program loop.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	FBF_TRIG	F_TRIG	Auto	No	0	..	F_TRIG (Falling edge trigger function block)

LD example (PTP115A000, F_TRIG_LD)



IL example

```
CAL FBF_TRIG (* Call the F_TRIG function block *)
LD Di00M00
ST FBF_TRIG.CLK (* Transfer the digital input to the function block clock *)
LD FBF_TRIG.Q
ST Do00M00 (* On the falling edge of digital input the output is set *)
```

ST example

```
FBF_TRIG(); (* Call the F_TRIG function block *)
FBF_TRIG.CLK:=Di00M00; (* Transfer the digital input to the function block clock *)
Do00M00:=FBF_TRIG.Q; (* On the falling edge of digital input the output is set *)
```

7.1.2 R_TRIG, Raising edge trigger

Type	Library
FB	ePLCStdLib_B000

This function block enables the **Q** output for a program loop on the rising edge of the **CLK** clock.



CLK (BOOL) Clock. On the rising edge signal, the Q output is turned on for a program loop.

Q (BOOL) Output. Active for a program loop on the rising edge of the CLK clock input.

Examples

On the rising edge of the digital **Di00M00**, digital output **Do00M00** became enabled for a program loop.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	FBR_TRIG	R_TRIG	Auto	No	0	..	R_TRIG (Raising edge trigger function block)

LD example (PTP115A100, R_TRIG_LD)



IL example

```
CAL FBR_TRIG (* Call the R_TRIG function block *)
LD Di00M00
ST FBR_TRIG.CLK (* Transfer the digital input to the function block clock *)
LD FBR_TRIG.Q
ST Do00M00 (* On the raising edge of digital input the output is set *)
```

ST example

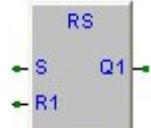
```
FBR_TRIG(); (* Call the R_TRIG function block *)
FBR_TRIG.CLK:=Di00M00; (* On the raising edge of digital input the counter count up *)
Do00M00:=FBR_TRIG.Q; (* On the raising edge of digital input the output is set *)
```

7.1.3 RS, Reset/Set flip flop

Type	Library
FB	ePLCStdLib_B000

This function block, on activation of the **S** command, set the **Q1** output that remains active even when the command is deactivated. To deactivate the output, the **R1** reset must be activated.

The R1 reset command has priority on S set command.



S (BOOL) Set. On activation of the signal, the Q1 output is activated and remains active even when the command is deactivated.

R1 (BOOL) Reset. On activation of the signal, the Q1 output is deactivated. Reset has priority on Set.

Q1 (BOOL) Output. Activated or deactivated according to the S set command and R1 reset command.

Examples

On activation of the **Di00M00** digital input, the **Do00M00** digital output becomes active even when the digital input is deactivated. To deactivate the digital output, you must activate the **Di01M00** digital input.

Note! The **Di01M00** digital input has priority on **Di00M00**.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	FBRS	RS	Auto	No	0	..	RS (Reset/Set function block)

LD example (PTP115A100, RS_LD)



IL example

```

CAL FBRS (* Call the RS function block *)
LD Di00M00
ST FBRS.S (* Transfer the digital input to the set command *)
LD Di01M00
ST FBRS.R1 (* Transfer the digital input to the reset command *)
LD FBRS.Q1
ST Do00M00 (* The function block output is copied to digital output *)

```

ST example

```

FBRS(); (* Call the RS function block *)
FBRS.S:=Di00M00; (* Transfer the digital input to the set command *)
FBRS.R1:=Di01M00; (* Transfer the digital input to the reset command *)
Do00M00:=FBRS.Q1; (* The function block output is copied to digital output *)

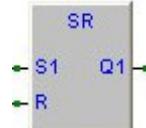
```

7.1.4 SR, Set/Reset flip flop

Type	Library
FB	ePLCStdLib_B000

This function block, on activation of the **S1** command, set the **Q1** output that remains active even when the command is deactivated. To deactivate the output, the **R** reset must be activated.

The S1 set command has priority on R reset command.



S1 (BOOL) Set. On activation of the signal, the Q1 output is activated and remains active even when the command is deactivated. Set has priority on Reset.

R (BOOL) Reset. On activation of the signal, the Q1 output is deactivated.

Q1 (BOOL) Output. Activated or deactivated according to the S set command and R1 reset command.

Examples

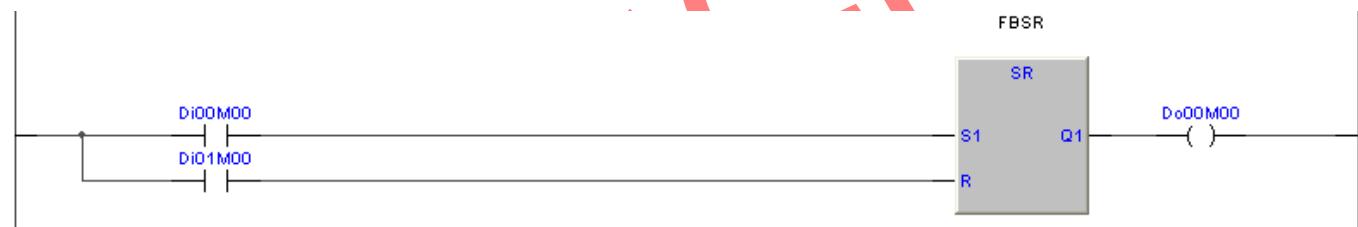
On activation of the **Di00M00** digital input, the **Do00M00** digital output becomes active and remains active even when the digital input is deactivated. To deactivate the digital output you must activate the **Di01M00** digital input.

Note! The **Di00M00** digital input has priority on **Di01M00**.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	FBSR	SR	Auto	No	0	..	SR (Set/Reset function block)

LD example (PTP115A100, SR_LD)



IL example

```

CAL FBSR (* Call the SR function block *)
LD Di00M00
ST FBRS.S1 (* Transfer the digital input to the set command *)
LD Di01M00
ST FBRS.R (* Transfer the digital input to the reset command *)
LD FBRS.Q1
ST Do00M00 (* The function block output is copied to digital output *)

```

ST example

```

FBSR(); (* Call the SR function block *)
FBRS.S1:=Di00M00; (* Transfer the digital input to the set command *)
FBRS.R:=Di01M00; (* Transfer the digital input to the reset command *)
Do00M00:=FBRS.Q1; (* The function block output is copied to digital output *)

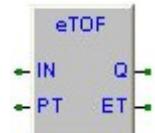
```

7.2 Functions and FBs for timers management

7.2.1 eTOF, Timer Off

Type	Library
FB	ePLCStdLib_B000

This function block performs the timing on deactivation. Activating the **IN** input, the **Q** output is activated immediately and the time on **ET** output is reset. Deactivating the IN input, starts the count and after the time set **PT** expressed in mS, you disable the Q output. On ET output, is back out the time elapsed since the deactivation expressed in mS.



- IN (BOOL)** Timer input. Activated, the Q output is activated immediately and the ET time output is reset. Deactivated, the count starts and after the time set in PT, the Q output is deactivated.
- PT (UDINT)** Preset time. Defines the delay time from input IN deactivation and Q output deactivation, expressed in mS.
- Q (BOOL)** Timer output. It is activated on IN input activation, and it is deactivated after the time set by PT from IN input deactivation.
- ET (UDINT)** Timer time. It is reset on IN input activation and it starts counting from the IN input deactivation. Reached the set time in PT, it stops counting. It is expressed in mS.

Examples

The timer is preset to 1 second (1000 mS). Activating the **Di00M00** digital input, the **Do00M00** digital output is activated immediately and the time value in the **VarOut** variable is clear. The timer is preset to 1 second (1000 mS).

Deactivating the **Di00M00** digital input, the timer starts counting and the value of elapsed time is copied in the **VarOut** variable. After the interval, the **Do00M00** digital output is disabled.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	FBeTOF	eTOF	Auto	No	0	..	eTOF (Timer Off function block)
2	OutValue	UDINT	Auto	No	0	..	Output value

LD example (PTP115A100, eTOF_LD)



IL example

```

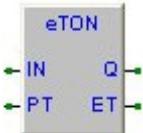
CAL FBeTOF (* Call the eTOF function block *)
LD Di00M00
ST FBeTOF.IN (* Transfer the digital input to timer input *)
LD 1000
ST FBeTOF.PT (* Set the delay time *)
LD FBeTOF.Q
ST Do00M00 (* When time is passed the digital output is set *)
LD FBeTOF.ET
ST OutValue (* The counting time is copied to variable *)

```

7.2.2 eTON, Timer On

Type	Library
FB	ePLCStdLib_B000

This function block performs the timing on the activation. By activating the **IN** input, the count starts and after the time set in **PT** expressed in mS, activates the **Q** output. On **ET** is back out the elapsed time since input activation expressed in mS. Turning off the **IN** input, the **Q** output deactivates immediately and the time value in **ET** is reset.



- IN** (BOOL) Timer input. Activating it, the count start and after the time set in PT, the Q output is activated. Disabling it, the Q output is deactivated immediately and the ET output value is reset.
- PT** (UDINT) Preset time. Defines the time delay from IN input activation to Q output activation, expressed in mS.
- Q** (BOOL) Timer output. It is activated after the time set in PT from the IN input activation and deactivated when IN becomes deactivated.
- ET** (UDINT) Timer time. Start counting from the IN input activation. Reached time set in PT, the counting stop. It is clear on IN input deactivation.

Examples

On activation of the **Di00M00** digital input, after 1 S (1000 ms) the **Do00M00** digital output is activated. Deactivating the **Di00M00** digital input, the **Do00M00** digital output is deactivated immediately.

Defining variables

	Name	Type	Address	Array	Initvalue	Attribute	Description
1	FBeTON	eTON	Auto	No	0	..	eTON (Timer On function block)
2	OutValue	UDINT	Auto	No	0	..	Output value

LD example (PTP115A100, eTON_LD)



IL example

```

CAL FBeTON (* Call the eTON function block *)
LD Di00M00
ST FBeTON.IN (* Transfer the digital input to timer input *)

LD 1000
ST FBeTON.PT (* Set the delay time *)

LD FBeTON.Q
ST Do00M00 (* When time is passed the digital output is set *)

LD FBeTON.ET
ST OutValue (* The counting time is copied to variable *)

```

ST example

```

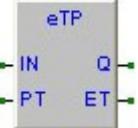
FBeTON(PT:=1000); (* Call the eTON function block *)
FBeTON.IN:=Di00M00; (* Transfer the digital input to timer input *)
OutValue:=FBeTON.ET; (* The counting time is copied to variable *)
Do00M00:=F_eTON.Q; (* When time is passed the digital output is set *)

```

7.2.3 eTP, Timer pulse

Type	Library
FB	ePLCStdLib_B000

This function block performs the timing of trigger pulse. Activating the **IN** input, the **Q** output becomes active and in **ET** there is the elapsed time (in ms) from impulse activation. Reached the **PT** set time (in ms), regardless of the status of **IN**, the **Q** output is reset, while the time on **ET** is reset only if **IN** input becomes deactivate.



- IN** (BOOL) Timer input. Activating it, the Q output is activated and the count begins. After the time set in PT regardless of the status of IN input, the Q output becomes deactivate.
- PT** (UDINT) Preset time. Defines the activation time of Q output, expressed in mS.
- Q** (BOOL) Timer output. It is activated on IN input activation for the time defined in PT.
- ET** (UDINT) Timer time. Start counting from the activation of IN input. Reached the PT set time, it stops counting. It is expressed in mS.

Examples

The timer is preset to 5 seconds (5000 mS). Activating the **Di00M00** digital input, the **Do00M00** digital output is activated immediately and the time value in the **VarOut** variable starts counting. Reached the set time (5 seconds), the **Do00M00** output is deactivated while the value of time in the **VarOut** variable remains locked on the preset value (5000 ms) until the deactivation of **Di00M00**.

Deactivating the **Di00M00** digital input during the timing count, did not affect either **Do00M00** output status, nor the value of the **VarOut** variable.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	FBeTP	eTP	Auto	No	0	..	eTP (Timer pulse function block)
2	OutValue	UDINT	Auto	No	0	..	Output value

LD example (PTP115A100, eTP_LD)



IL example

```

CAL FBeTP (* Call the eTP function block *)
LD Di00M00
ST FBeTP.IN (* Transfer the digital input to timer input *)
LD 5000
ST FBeTP.PT (* Set the delay time *)
LD FBeTP.Q
ST Do00M00 (* When time is passed the digital output is set *)
LD FBeTP.ET
ST OutValue (* The counting time is copied to variable *)

```

ST example

```

FBeTP(PT:=5000); (* Call the eTP function block *)
FBeTP.IN:=Di00M00; (* Transfer the digital input to timer input *)
OutValue:=FBeTP.ET; (* The counting time is copied to variable *)
Do00M00:=FBeTP.Q; (* When time is passed the digital output is set *)

```

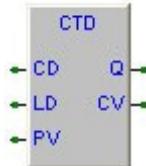
7.3 Functions and FBs for counters management

7.3.1 CTD, Counter Down

Type	Library
FB	ePLCStdLib_B000

This function block performs the management of a counter down. Acting on **LD** load input is possible at any time to transfer the **PV** preset value on the **CV** counter. Each rising edge of the **CD** input, the **CV** counter value is decreased. When the value reaches 0, the **Q** output is activated and the counting stops. Only by acting on the **LD** load input is possible to preset the counter and to restart the counter again.

The LD load input has priority over the CD decrease input.



- CD (BOOL)** Decrement counter command. For each rising edge, the CV counter value is decreased.
- LD (BOOL)** Load command. Activating the input the PV preset value is transferred to the CV counter value.
- PV (INT)** Preset value. Activating the LD load input, the value is transferred to the CV.
- Q (BOOL)** Counter output. Activated when the counter value reaches 0.
- CV (INT)** Counter value. When the value reaches 0, the output Q is set and the counter does not decrease any more.

Examples

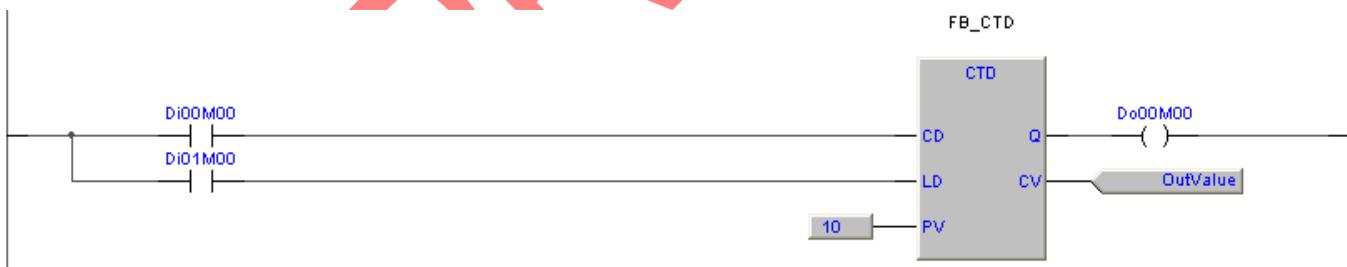
The counter is preset to 10 and its output value (CV) is copied in **OutValue**. Activating the **Di01M00** digital input the counter is preset and its CV output value is set at 10, also resetting the Q output.

On the **Di00M00** rising edge, the digital counter is decremented by 1. When the count reaches zero, the count stops and the Q output of the counter is activated so also the **Do00M00** digital output. To restart the count must activate the **Di01M00** digital input that preset the counter.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	FB_CTD	CTD	Auto	No	0	..	CTD (Counter Down function block)
2	OutValue	INT	Auto	No	0	..	Output value

LD example (PTP115A100, CTD_LD)



IL example

```
CAL FB_CTD (* Call the CTD function block *)
LD 10
ST FB_CTD.PV (* Preset value *)

LD Di00M01
ST FB_CTD.CD (* On the raising edge of digital input the counter count down *)

LD Di01M00
ST FB_CTD.LD (* If the digital input is set the PV value is loaded *)

LD FB_CTD.Q
ST Do00M00 (* If the counter value is 0 the digital output is set *)

LD FB_CTD.CV
ST OutValue (* The counter value is copied to the variable *)
```

ST example

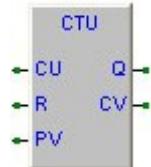
```
FB_CTD(PV:=10); (* Call the CTD function block and preset counter *)
FB_CTD.CD:=Di00M00; (* On the raising edge of digital input the counter count down *)
FB_CTD.LD:=Di01M00; (* If the digital input is set the PV value is loaded *)
Do00M00:=FB_CTD.Q; (* If the counter value is 0 the digital output is set *)
OutValue:=FB_CTD.CV; (* The counter value is copied to the variable *)
```

DEPRECATED

7.3.2 CTU, Counter Up

Type	Library
FB	ePLCStdLib_B000

This function block performs the operation of an incremental counter. Acting on **R** reset input, at any time you can reset the **CV** counter value. Each rising edge of **CU** input, the counter value **CV** is incremented. When the value of the counter **CV** reaches the preset value, defined on **PV**, output **Q** is activated and the counting stops. Only with the **R** reset input the counter will be reset and you will be able to restart the counter again.



The R reset input is priority on CU.

- CU (BOOL)** Increment counter command. For each rising edge, the counter value CV is increased.
- R (BOOL)** Reset command. Activating the input, the value of the counter is reset.
- PV (INT)** Preset value. When the value of the CV counter reaches this value, the Q output is set and the counter no longer increments.
- Q (BOOL)** Counter output. Activated if the CV value of the counter reaches the value defined in PS.
- CV (INT)** Value counter. When it reaches the PV preset value, the Q output is set and the counter does not increment anymore.

Examples

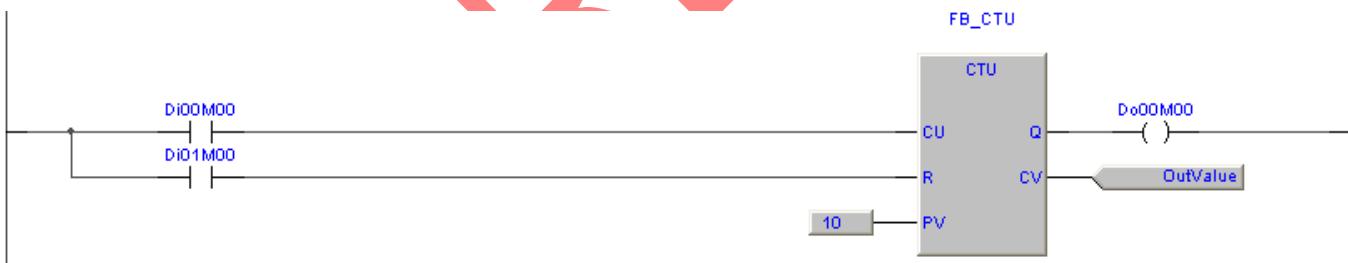
The counter is preset to 10 and its CV output value is copied in the **OutValue** variable. Activating the **Di01M00** digital input, the counter is reset and its CV output value is set to 0, also resetting the Q output.

On the **Di00M00** rising edge, the counter is incremented by 1. When the count value reaches the preset value, the count stops and the Q output of the counter is activated, so also the **Do00M00** digital output is activated. To restart the count must activate the **Di01M00** digital input which resets the counter.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	FB_CTU	CTU	Auto	No	0	..	CTU (Counter Up function block)
2	OutValue	INT	Auto	No	0	..	Output value

LD example (PTP115A100, CTU_LD)



IL example

```

CAL FBCTU (* Call the CTU function block *)
LD 10
ST FBCTU.PV (* Preset counter *)

LD Di00M00
ST FB_CTU.CU (* On the raising edge of digital input the counter count up *)

LD Di01M00
ST FB_CTU.R (* If the digital input is set the counter is reset *)

LD FBCTU.Q
ST Do00M00 (* If the counter value has reached the preset the digital output is set *)

LD FB_CTU.CV
ST OutValue (* The counter value is copied to the variable *)

```

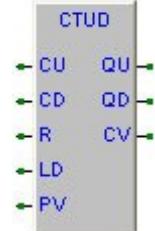
7.3.3 CTUD, Counter Up/Down

Type	Library
FB	ePLCStdLib_B000

This function block performs the operation of a counter up and down. Acting on **R** reset input, at any time you can reset the **CV** counter value. Acting on **LD** input, is possible at any time to transfer the **PV** preset value on the **CV** counter value.

Each rising edge of the **CU** input, the **CV** counter value is incremented. When the value of the **CV** counter reaches the preset value, defined on **PV**, the **Q** output is set and the counting stops. Only with the **R** reset input the counter will be reset and you will be able to restart the counter again.

Each rising edge of the **CD** input, the **CV** counter value is decreased. When the value reaches 0, the **Q** output is set and the counting stops. Only by acting on the **LD** input is possible to preset the counter and to restart the counter again.



- CU** (BOOL) Increment counter command. For each rising edge of the counter, the CV value is increased.
- CD** (BOOL) Decrement counter command. For each rising edge, the counter value of CV decreases.
- R** (BOOL) Reset command. Activating the input, the value of the counter is reset.
- LD** (BOOL) Load command. Activating the input, the PV preset value is transferred to the CV counter value.
- PV** (INT) Preset value. When the value of the CV counter reaches this value, the Q output is set and the counter no longer increments.
- QU** (BOOL) Output up counter. Activated if the value of the CV counter reaches the value defined in PS presets.
- QD** (BOOL) Down counter output. Activated if the value of CV counter reaches 0.
- CV** (INT) Counter value. When it reaches the PV preset value, the Q output is activated and the counter does not increment anymore.

Examples

The counter is preset to 10 and its CV output value is copied in the **VarOut** variable. Activating the **Di02M00** digital input, the counter is reset and its CV output value is set to 0, also resetting the QU output. Activating the **Di03M00** digital input the counter is preset and its CV output value is set at 10, also resetting the QD output.

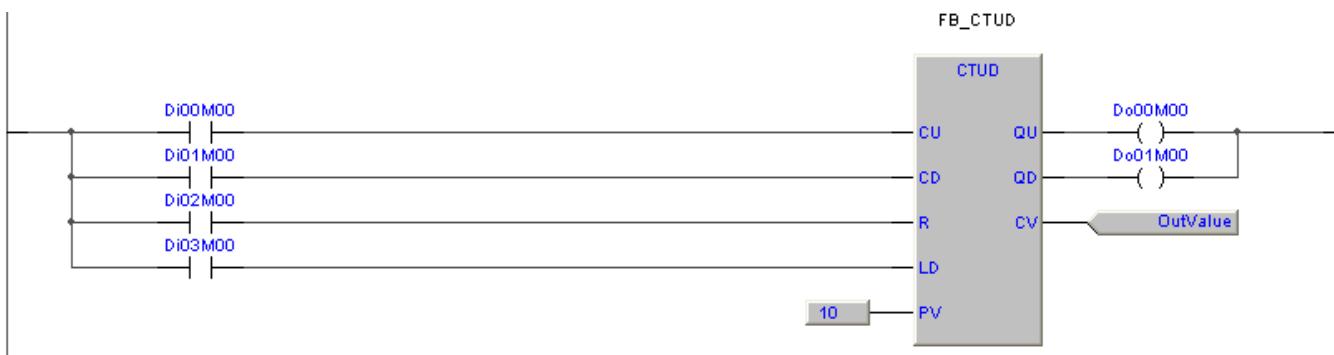
On the rising edge of the **Di00M00** digital input, the counter is incremented by 1. When the count value reaches the preset value, the count stops and the QU output is activated, so also **Do00M00** digital output. To restart the count must activate the **Di02M00** digital input which resets the counter.

On the rising edge of the **Di01M00** digital input, the counter is decremented by 1. When the count reaches zero, the count stops and activates the QD output that activates the **Do01M00** digital output also. To restart the count must activate the **Di03M00** digital input that preset the counter.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	FB_CTUD	CTUD	Auto	No	0	..	CTUD (Counter Up/Down function block)
2	OutValue	INT	Auto	No	0	..	Output value

LD example (PTP115A100, CTUD_LD)



IL example

```

CAL FB_CTUD (* Call the CTUD function block *)
LD 10
ST FB_CTUD.PV (* Preset value *)

LD Di00M00
ST FB_CTUD.CU (* On the raising edge of digital input the counter count up *)

LD Di01M00
ST FB_CTUD.CD (* On the raising edge of digital input the counter count down *)

LD Di02M00
ST FB_CTUD.R (* If the digital input is set the counter is reset *)

LD Di03M00
ST FB_CTUD.LD (* If the digital input is set the PV value is loaded *)

LD FB_CTUD.QU
ST Do00M00 (* If the counter value has reached the preset the digital output is set *)

LD FB_CTUD.QD
ST Do01M00 (* If the counter value is 0 the digital output is set *)

LD FB_CTUD.CV
ST OutValue (* The counter value is copied to the variable *)

```

ST example

```

FB_CTUD(PV:=10); (* Call the CTD function block and preset counter *)

FB_CTUD.CU:=Di00M00; (* On the raising edge of digital input the counter count up *)
FB_CTUD.CD:=Di01M00; (* On the raising edge of digital input the counter count down *)
FB_CTUD.R:=Di02M00; (* If the digital input is set the counter is reset *)
FB_CTUD.LD:=Di03M00; (* If the digital input is set the PV value is loaded *)
Do00M00:=FB_CTUD.QU; (* If the counter value has reached the preset the digital output is set *)
Do01M00:=FB_CTUD.QD; (* If the counter value is 0 the digital output is set *)
OutValue:=FBCTUD.CV; (* The counter value is copied to the variable *)

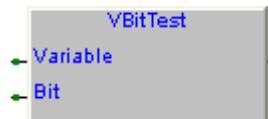
```

7.4 Functions and FBs for data conversion

7.4.1 VBitTest, Variable bit test

Type	Library
Function	eLLabUtyLib_C030

This function executes the test of a bit in a variable.



Function parameters:

Variable (UDINT) Variable in which to test the bit.

Bit (USINT) Number of bit to test (Range from 0 to 31).

Return value:

(BOOL) Selected bit status.

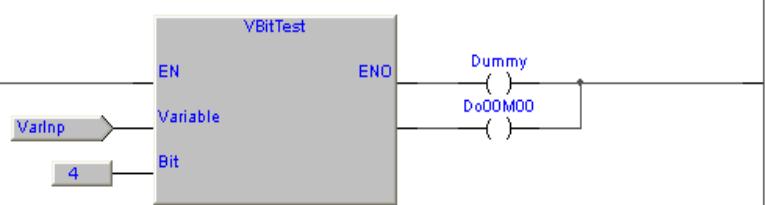
Examples

The state of bit 4 of the **VarInp** variable is transferred to **Do00M00** digital output.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Dummy	BOOL	Auto	No	FALSE	..	Dummy variable
2	VarInp	UDINT	Auto	No	0	..	Variable input

LD example



IL example

```
LD VarInp (* Variable input *)
VBitTest 4 (* Variable bit test *)
ST Do00M00 (* Transfer bit status to digital output *)
```

ST example

```
Do00M00:=VBitTest(VarInp, 4); (* Variable bit test *)
```

7.4.2 VBitSet, Variable bit set

Type	Library
Function	eLLabUtyLib_C030

This function executes the set of a bit in a variable.



Function parameters:

Value (BOOL) Bit value to set.

Variable (UDINT) Variable in which to set the bit.

Bit (USINT) Number of bit to set (Range from 0 to 31).

Return value:

(UDINT) Variable value after set of bit.

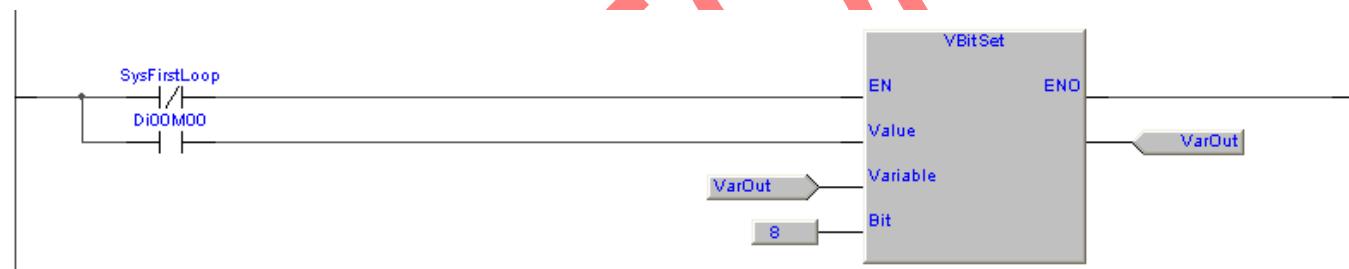
Examples

The state of **Di00M00** digital input is transferred in bit number 8 of the **VarOut** variable.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	VarOut	UDINT	Auto	No	0	..	Variable output

LD example



IL example

```

LD Di00M00 (* Variable input *)
VBitSet VarOut, 8 (* Variable bit set *)
ST VarOut (* Transfer result to variable *)
  
```

ST example

```

VarOut:=VBitSet(Di00M00, VarOut, 8); (* Variable bit set *)
  
```

7.4.3 BitToByte, Bit to byte conversion

Type	Library
FB	eLLabUtyLib_C030

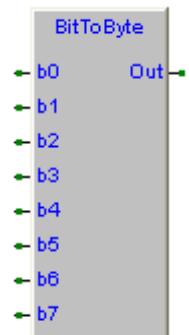
This function block allows you to convert 8 **BOOL** variables in a **BYTE** variable.

b0 (BOOL) Bit 0 of **Out** Byte.

... ...

b7 (BOOL) Bit 7 of **Out** Byte.

Out (BYTE) Input bit conversion result.



Examples

The 8 digital input of module 0 are transferred into the **VarOut** variable. Activating the **Di00M00** digital input, the **VarOut** variable value will be 1. Activating the **Di01M00** digital input **VarOut** will be 2, and so on until activating the **Di07M00** where **VarOut** assumes the value 128. Activating multiple inputs simultaneously, the **VarOut** variable will take the sum of activated inputs.

For simplicity in the IL and ST examples are not reported all the bits.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	FBData	BitToByte	Auto	No	0	..	FB Bit to byte data
2	VarOut	UDINT	Auto	No	0	..	Variable output

LD example (PTP114A100, LD_BitToByte)



IL example (PTP114A100, IL_BitToByte)

```

LD  Di00M00
ST  FBData.b0 (* Transfer digital input to input bit *)

LD  Di07M00
ST  FBData.b7 (* Transfer digital input to input bit *)

CAL FBData (* Call the BitToByte function block *)

LD  FBData.Out
ST  VarOut (* Transfer the result to variable *)

```

ST example (PTP114A100, ST_BitToByte)

```
FBDData.b0:=Di00M00; (* Transfer digital input to input bit *)
FBDData.b7:=Di07M00; (* Transfer digital input to input bit *)

FBDData(); (* Call the BitToByte function block *)

VarOut:=FBDData.Out; (* Transfer the result to variable *)
```

DEPRECATED

7.4.4 ByteToBit, Byte to bit conversion

Type	Library
FB	eLLabUtyLib_C030

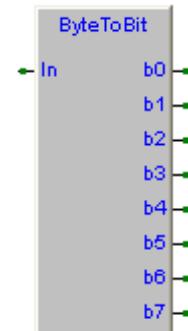
This function block allows you to convert a **BYTE** variable in 8 **BOOL** variables.

In (BYTE) Byte value to convert

b0 (BOOL) Bit 0 of **In**.

...

b7 (BOOL) Bit 7 of **In**.



Examples

The state of bit 0 of the **VarInp** variable is transferred to **Do00M00** digital output. The state of bit 1 of the **VarInp** variable is transferred to **Do01M00** digital output and so on until the status of bit 7 that is transferred to **Do07M00** digital output.

For simplicity in the IL and ST examples are not reported all the bits.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	VarInp	USINT	Auto	No	0	..	Variable input
2	FBData	ByteToBit	Auto	No	0	..	FB Byte to bit data
3	Dummy	BOOL	Auto	No	FALSE	..	Dummy variable

LD example (PTP114A100, LD_ByteToBit)



IL example (PTP114A100, IL_ByteToBit)

```

LD  VarInp
ST  FBData.In (* Transfer the variable to input *)
CAL FBData (* Call the ByteToBit function block *)

LD  FBData.b0
ST  Di00M00 (* Transfer output bit to digital output *)

```

ST example (PTP114A100, ST_ByteToBit)

```

FBData(In:=VarInp); (* Call the ByteToBit function block *)

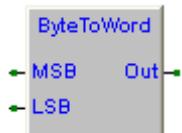
Do00M00:=FBData.b0; (* Transfer output bit to digital output *)
Do01M00:=FBData.b1; (* Transfer output bit to digital output *)

```

7.4.5 ByteToWord, Byte to word conversion

Type	Library
FB	eLLabUtyLib_C030

This function block allows you to convert two **BYTE** variables in a single **WORD** variable.



MSB (BYTE) MSB of the Out output value

LSB (BYTE) LSB of the Out output value

Out (WORD) Output value

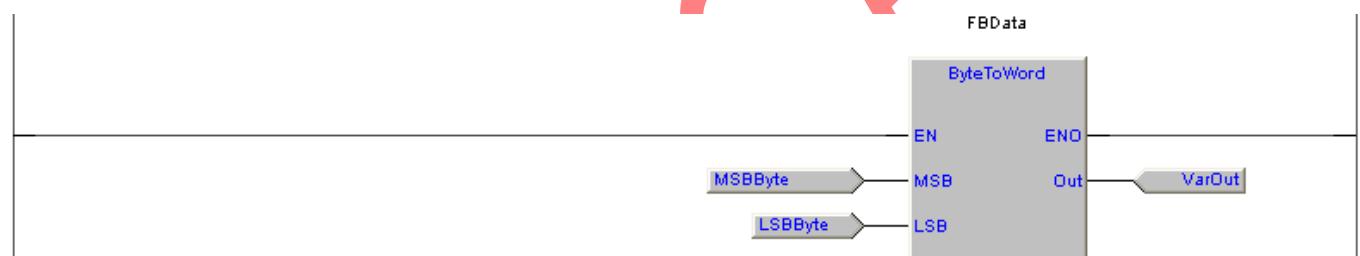
Examples

The **MSBByte** and **LSBByte** variables are joined in the **VarOut** output variable.

Defining variables

	Name	Type	Address	Array	Initvalue	Attribute	Description
1	FBData	ByteToWord	Auto	No	0	..	FB Byte to word data
2	LSBByte	BYTE	Auto	No	0	..	Valore byte MSB
3	MSBByte	BYTE	Auto	No	0	..	Valore byte LSB
4	VarOut	WORD	Auto	No	0	..	Variable output

LD example (PTP114A100, LD_ByteToWord)



IL example (PTP114A100, IL_ByteToWord)

```

LD  MSBByte
ST  FBData.MSB (* Transfer the MSB variable to input *)

LD  LSBByte
ST  FBData.LSB (* Transfer the LSB variable to input *)

CAL FBData (* Call the ByteToWord function block *)

LD  FBData.Out
ST  VarOut (* Transfer output to variable *)

```

ST example (PTP114A100, ST_ByteToWord)

```

FBData.MSB:=MSBByte; (* Transfer the MSB variable to input *)
FBData.LSB:=LSBByte; (* Transfer the LSB variable to input *)

FBData(); (* Call the ByteToWord function block *)

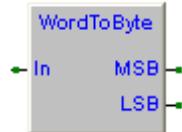
VarOut:=FBData.Out; (* Transfer output to variable *)

```

7.4.6 WordToByte, Word to byte conversion

Type	Library
FB	eLLabUtyLib_C030

This function block allows you to split a **WORD** variable into two **BYTE** variables.



IN (WORD) Variable to convert.

MSB (BYTE) MSB of input value.

LSB (BYTE) LSB of input value.

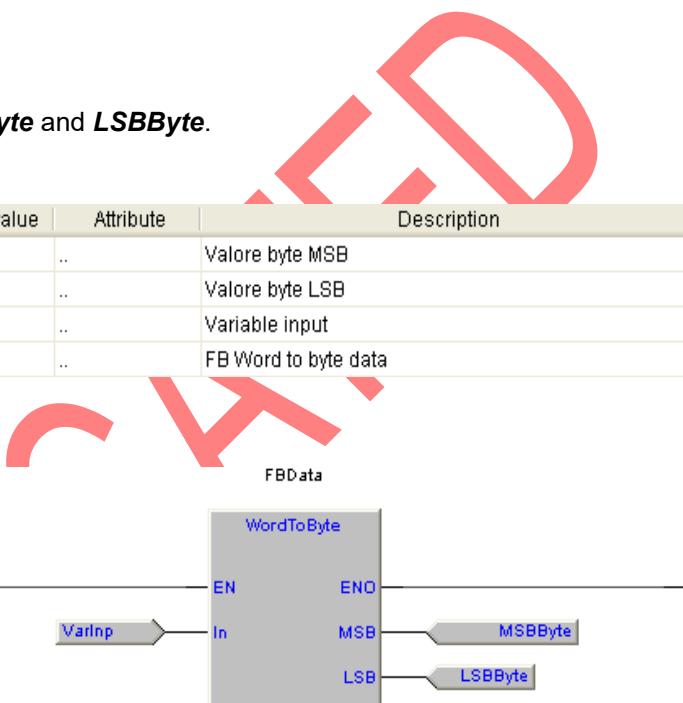
Examples

The **VarInp** variable is splitted into two variables: **MSBByte** and **LSBByte**.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	LSBByte	BYTE	Auto	No	0	..	Valore byte MSB
2	MSBByte	BYTE	Auto	No	0	..	Valore byte LSB
3	VarInp	WORD	Auto	No	0	..	Variable input
4	FBData	WordToByte	Auto	No	0	..	FB Word to byte data

LD example (PTP114A100, LD_WordToByte)



IL example (PTP114A100, IL_WordToByte)

```

LD  VarInp
ST  FBData.In (* Transfer the variable to input *)

CAL FBData (* Call the WordToByte function block *)

LD  FBData.MSB
ST  MSBByte (* Transfer the MSB output to variable *)

LD  FBData.LSB
ST  LSBByte (* Transfer the LSB output to variable *)

```

ST example (PTP114A100, ST_WordToByte)

```

FBData.In:=VarInp; (* Transfer the variable to input *)

FBData(); (* Call the WordToByte function block *)

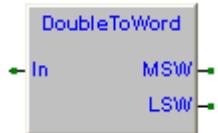
MSBByte:=FBData.MSB; (* Transfer the MSB output to variable *)
LSBByte:=FBData.LSB; (* Transfer the LSB output to variable *)

```

7.4.7 DoubleToWord, Double to word conversion

Type	Library
FB	eLLabUtyLib_C030

This function block allows you to split a **DWORD** variable into two **WORD** variables.



IN (DWORD) Variable to convert.

MSW (WORD) MSW of input value.

LSW (WORD) LSW of input value.

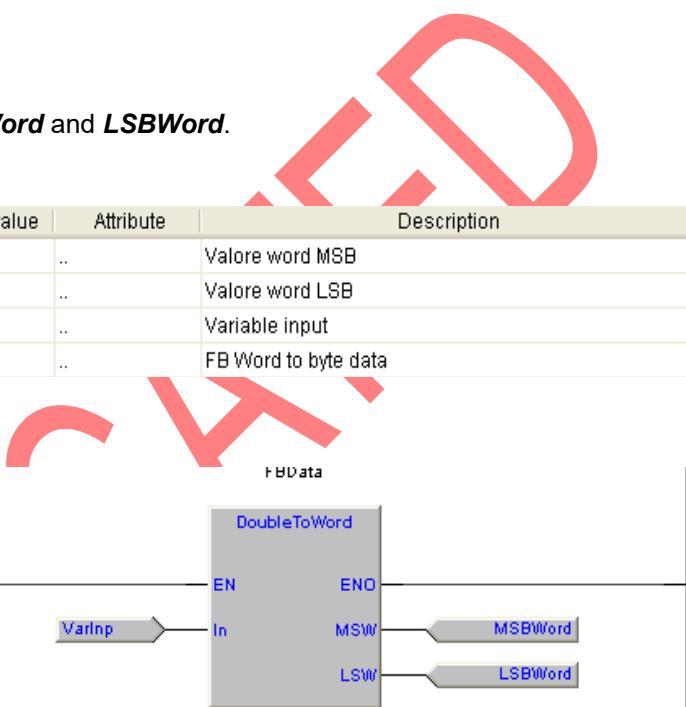
Examples

The **VarInp** variable is splitted into two variables: **MSBWord** and **LSBWord**.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	LSBWord	WORD	Auto	No	0	..	Valore word MSB
2	MSBWord	WORD	Auto	No	0	..	Valore word LSB
3	VarInp	DWORD	Auto	No	0	..	Variable input
4	FBDData	DoubleToWord	Auto	No	0	..	FB Word to byte data

LD example (PTP114A100, LD_DoubleToWord)



IL example (PTP114A100, IL_DoubleToWord)

```

LD VarInp
ST FBDData.In (* Transfer the variable to input *)

CAL FBDData (* Call the DoubleToWord function block *)

LD FBDData.MSW
ST MSBWord (* Transfer the MSW output to variable *)

LD FBDData.LSW
ST LSBWord (* Transfer the LSW output to variable *)

```

ST example (PTP114A100, ST_DoubleToWord)

```

FBDData.In:=VarInp; (* Transfer the variable to input*)

FBDData(); (* Call the DoubleToWord function block*)

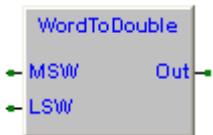
MSBWord:=FBDData.MSW; (* Transfer the MSW output to variable*)
LSBWord:=FBDData.LSW; (* Transfer the LSW output to variable*)

```

7.4.8 WordToDouble, Word to double conversion

Type	Library
FB	eLLabUtyLib_C030

This function block allows you to join two **WORD** variables in a single **DWORD** variable.



MSW (WORD) MSW of output value

LSW (WORD) LSW of output value

Out (DWORD) Output value

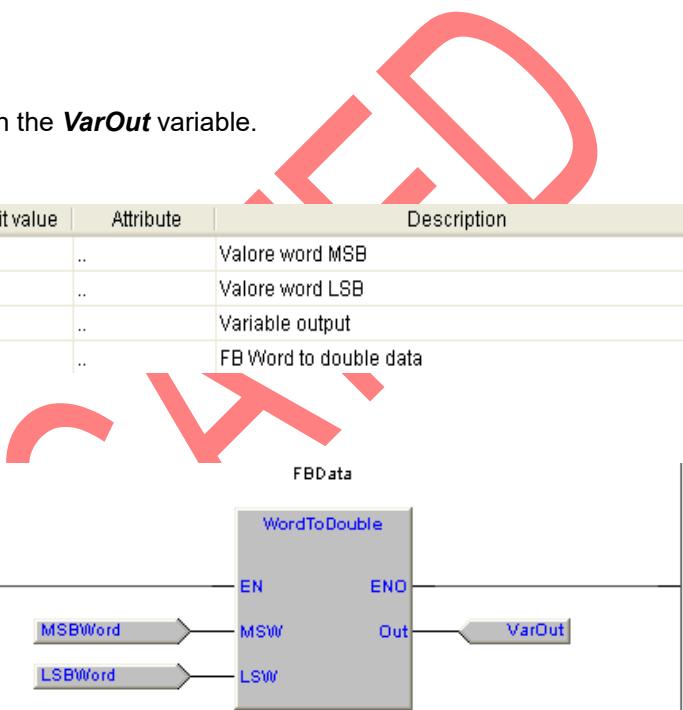
Examples

The two variables **MSBWord** and **LSBWord** are joined in the **VarOut** variable.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	LSBWord	WORD	Auto	No	0	..	Valore word MSB
2	MSBWord	WORD	Auto	No	0	..	Valore word LSB
3	VarOut	DWORD	Auto	No	0	..	Variable output
4	FBDData	WordToDouble	Auto	No	0	..	FB Word to double data

LD example (PTP114A100, LD_WordToDouble)



IL example (PTP114A100, IL_WordToDouble)

```

LD  MSBWord
ST  FBDData.MSW (* Transfer the MSW variable to input *)

LD  LSBWord
ST  FBDData.LSW (* Transfer the LSW variable to input *)

CAL FBDData (* Call the WordToDouble function block *)

LD  FBDData.Out
ST  VarOut (* Transfer output to variable *)

```

ST example (PTP114A100, ST_WordToDouble)

```

FBDData.MSW:=MSBWord; (* Transfer the MSW variable to input *)
FBDData.LSW:=LSBWord; (* Transfer the LSW variable to input *)

FBDData(); (* Call the WordToDouble function block *)

VarOut:=FBDData.Out; (* Transfer output to variable *)

```

7.4.9 ToLower, Uppercase to lowercase letter conversion

Type	Library
Function	eLLabUtyLib_C030

This function converts an uppercase character in the corresponding lowercase character.

Function parameters:

Char (USINT) Character to convert.

Return value:

(USINT) Character in lowercase format.

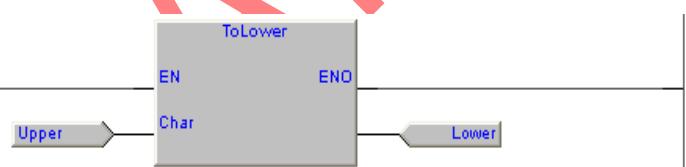
Examples

The **Upper** variable is converted to the corresponding lowercase character and transferred to **Lower** variable. The init value 16#41, which corresponds to the letter 'A', is converted to the value 16#61, which corresponds to the letter 'a'.

Defining variables

	Name	Type	Address	Array	Initvalue	Attribute	Description
1	Upper	USINT	Auto	No	16#41	..	Uppercase letter
2	Lower	USINT	Auto	No	0	..	Lowercase letter

LD example



IL example

```

LD  Upper (* Uppercase letter *)
ToLower (* Uppercase to lowercase letter conversion *)
ST  Lower (* Lowercase letter *)
  
```

ST example

```

Lower:=ToLower(Upper); (* Uppercase to lowercase letter conversion *)
  
```

7.4.10 ToUpper, Lowercase to uppercase letter conversion

Type	Library
Function	eLLabUtyLib_C030

This function converts a lowercase character in the corresponding uppercase character.



Function parameters:

Char (USINT) Character to convert.

Return value:

(USINT) Character in uppercase format.

Examples

The **Lower** variable is converted to the corresponding uppercase character and transferred to **Upper** variable. The init value 16#61, which corresponds to the letter 'a', is converted to the value 16#41, which corresponds to the letter 'A'.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Lower	USINT	Auto	No	16#61	..	Lowercase letter
2	Upper	USINT	Auto	No	0	..	Uppercase letter

LD example



IL example

```
LD Lower (* Lowercase letter *)
ToUpper (* Lowercase to uppercase letter conversion *)
ST Upper (* Uppercase letter *)
```

ST example

```
Upper:=ToUpper(Lower); (* Lowercase to uppercase letter conversion *)
```

7.4.11 LEArrayToVar, Little endian array to variable conversion

Type	Library
Function	eLLabUtyLib_C030

This function acquires the value from a little endian (MSB – LSB) array **Source** and transfers it to the destination variable **Destination**, according to the type defined in **Type** variable.



Function parameters:

Type (USINT) Variable type as defined in the [Variable types definition](#) table.

Destination (@USINT) Destination variable address.

Source (@USINT) Source array address.

Return value:

(BOOL) FALSE: Wrong variable type, TRUE: Conversion Ok.

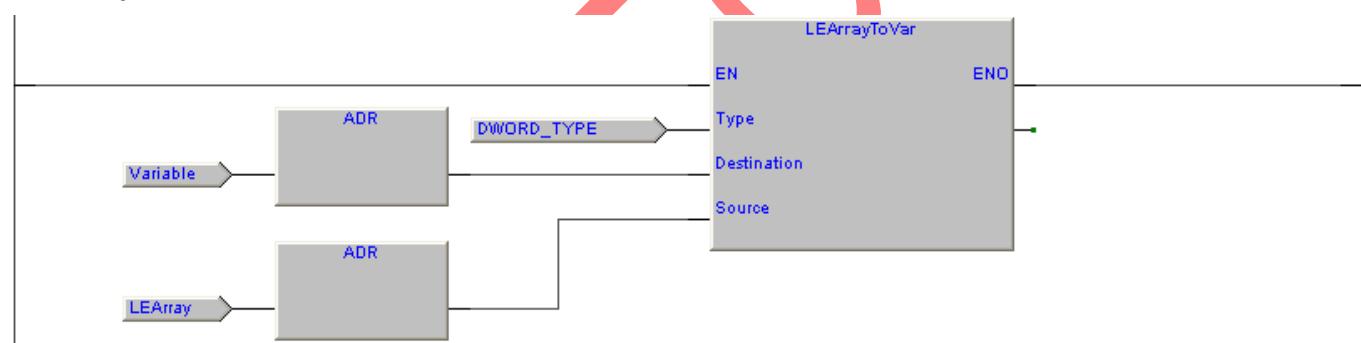
Examples

The value "1, 2, 3, 4" in the **LEArray** array is converted into **Variable**, after conversion the variable will have the value 16 # 01020304.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Variable	DWORD	Auto	No	0	..	Variable
2	LEArray	USINT	Auto	[0..3]	1,2,3,4	..	Little endian array

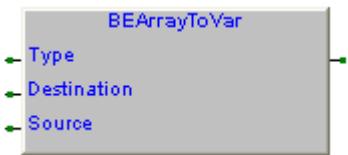
LD example



7.4.12 BEArrayToVar, Big endian array to variable conversion

Type	Library
Function	eLLabUtyLib_C030

This function acquires the value from a big endian (LSB – MSB) array **Source** and transfers it to the destination variable **Destination**, according to the type defined in **Type** variable.



Function parameters:

Type (USINT) Variable type as defined in the [Variable types definition](#) table.

Destination (@USINT) Destination variable address.

Source (@USINT) Source array address.

Return value:

(BOOL) FALSE: Wrong variable type, TRUE: Conversion Ok.

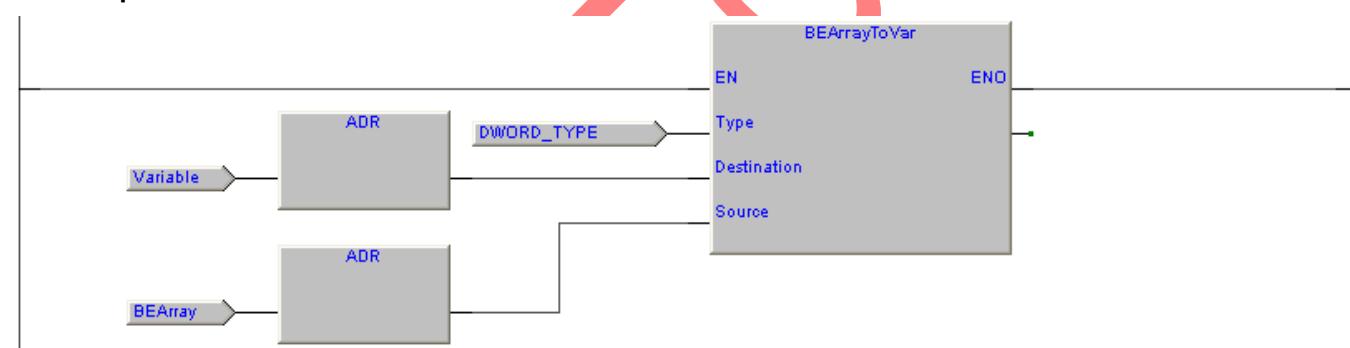
Examples

The value "4, 3, 2, 1" in the **BEArray** array is converted into **Variable**, after conversion the variable will have the value 16 # 01020304.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Variable	DWORD	Auto	No	0	..	Variable
2	BEArray	USINT	Auto	[0..3]	4,3,2,1	..	Big endian array

LD example



7.4.13 VarToLEArray, variable to little endian array conversion

Type	Library
Function	eLLabUtyLib_C030

This function transfers the value of a **Source** variable based on the **Type** defined, in an **Destination** array using the little endian format (MSB – LSB).



Function parameters:

Type (USINT) Variable type as defined in the [Variable types definition](#) table.

Destination (@USINT) Destination array address.

Source (@USINT) Source variable address.

Return value:

(BOOL) FALSE: Wrong variable type, TRUE: Conversion Ok.

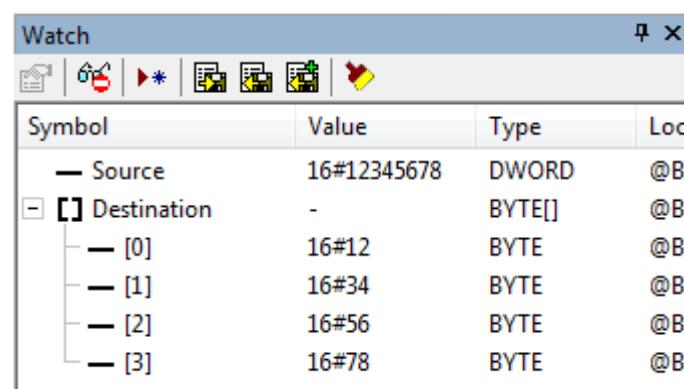
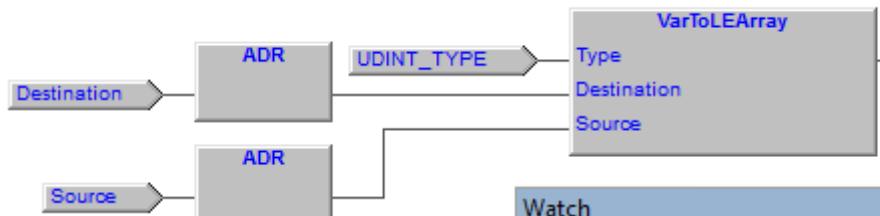
Examples

The value of the **Source** variable is converted to the little endian (MSB – LSB) **Destination** array. The example also reports the Watch window with the displayed values.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Source	DWORD	Auto	No	16#12345678	..	Source variable
2	Destination	BYTE	Auto	[0..3]		..	Destination array

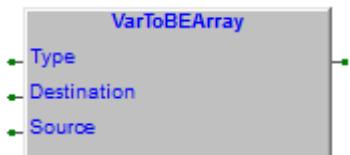
FBD example



7.4.14 VarToBEArray, variable to big endian array conversion

Type	Library
Function	eLLabUtyLib_C030

This function transfers the value of a **Source** variable based on the **Type** defined, in an **Destination** array using the big endian format (LSB – MSB).



Function parameters:

Type (USINT) Variable type as defined in the [Variable types definition](#) table.

Destination (@USINT) Destination array address.

Source (@USINT) Source variable address.

Return value:

(BOOL) FALSE: Wrong variable type, TRUE: Conversion Ok.

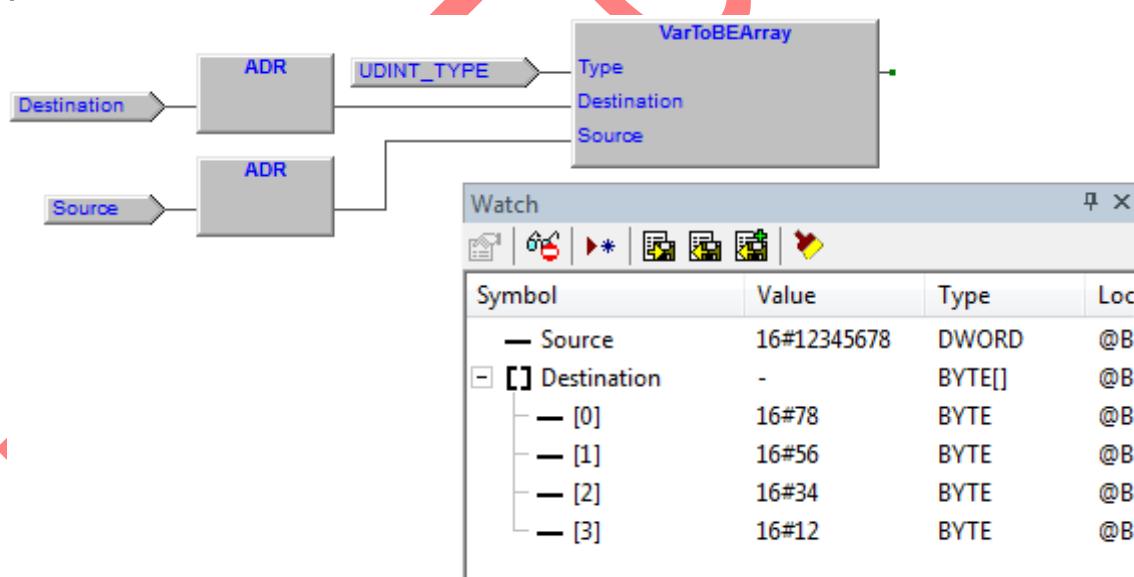
Examples

The value of the **Source** variable is converted to the big endian (LSB – MSB) **Destination** array. The example also reports the Watch window with the displayed values.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Source	DWORD	Auto	No	16#12345678	..	Source variable
2	Destination	BYTE	Auto	[0..3]		..	Destination array

FBD example



7.5 Functions and FBs of system utility

7.5.1 SysGetSysTime, get system time

Type	Library
Function	XTarget_07_0

This function returns the system time expressed in microseconds. Using the **Cmd** value, it is possible to define if you want to have the current system time (**Cmd:=TRUE**) or the one stored with the previous execution of the function (**Cmd:=FALSE**). The count starts from 0 and after 4294 seconds resets.



Function parameters:

Cmd (BOOL) Allows to select the returned time.

TRUE: The actual time is saved and returned.

FALSE: The saved time (saved from the previous call with **Cmd:=TRUE**) is returned.

Return value:

(UDINT) System time in μ S.

Examples

It is calculated the time that the digital input **Di00M00** remains in the activated condition..

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	StartTime	UDINT	Auto	No	0	..	Input raising time reference (μ S)
2	SetTime	UDINT	Auto	No	0	..	Input set time (μ S)
3	Pulse	BOOL	Auto	No	FALSE	..	Pulse flag

ST example (PTP116A000, ST_SysGetSysTime)

```
(* Check if input is activated. *)
IF (Di00M00 <> Pulse) THEN
    Pulse:=Di00M00; (* Pulse flag *)
    (* On input raising edge relate time is saved. *)
    IF (Di00M00) THEN StartTime:=SysGetSysTime(TRUE); END_IF;
    (* On input falling edge the set time is calculated. *)
    IF (NOT(Di00M00)) THEN SetTime:=SysGetSysTime(TRUE)-StartTime; END_IF;
END_IF;
```

Timeout calculation

Since the value of system time returned by the function an **UDINT** number incremented each μ S, and the maximum value to zero does overflow, you can not make direct comparisons with the reference time, but you should always make the difference.

The following example the **Timeout** is activated if the **Di00M00** input remains active for more than a second.

```
IF NOT(Di00M00) THEN TimeBf:=SysGetSysTime(TRUE);
ELSE IF ((SysGetSysTime(TRUE)-TimeBf) >= 1000000) THEN Timeout:=TRUE; END_IF;
END_IF;
```

The same example written in this way does not work properly.

```
IF NOT(Di00M00) THEN TimeBf:=SysGetSysTime(TRUE);
ELSE IF (SysGetSysTime(TRUE) >= (TimeBf+1000000)) THEN Timeout:=TRUE; END_IF;
END_IF;
```

A simple chronometer

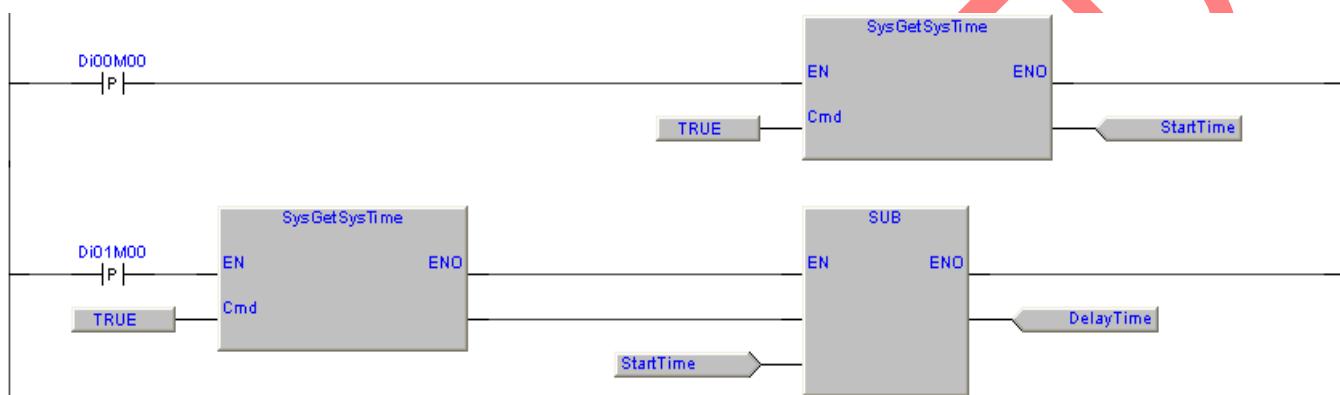
This example creates a simple chronometer to measure the time between a start command and a stop. For example, using two photocells one on the start line and one on the stop the of a path, it is possible to calculate the travel time expressed in μs .

Activating the ***Di00M00*** start input, the system time is saved in the ***StartTime*** variable. By activating the ***Di01M00*** stop input, the time elapsed between the start and stop is calculated. The calculated time is saved in the ***DelayTime*** variable.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	StartTime	UDINT	Auto	No	0	..	Start time (μs)
2	DelayTime	UDINT	Auto	No	0	..	Delay time (μs)

LD example (PTP119A000)



7.5.2 SysSetTaskLpTime, set task loop time

Type	Library
Function	XTarget_07_0

This feature allows you to set the execution time of PLC tasks. There are two settable tasks: the slow task (**ID_TASK_SLOW**) and the fast task (**ID_TASK_FAST**). Each of these tasks can be assigned an execution time.

If the set time is not included in the defined range or if the ratio of task execution times fast than the slow are not consistent, the function does not change the time of execution and returns **FALSE**. The following are the ranges of time defined for the various tasks.

ID_TASK_FAST Range from 100 µS to 10 mS

ID_TASK_SLOW Range from 1 to 100 mS

Function parameters:

TaskID (USINT) Identifies the task to which you want to set the execution time as defined in the [Task ID](#).

Time (UDINT) Indicates the value of task execution time expressed in µS.

Return value:

(BOOL) **TRUE**: If function successful.
FALSE: If an error occurs running, such as incorrect parameters.

Error codes

If an error occurs the function returns **FALSE** and [SysGetLastError](#) can detect the error code.

9948990 Not yet implemented in the simulator.

Examples

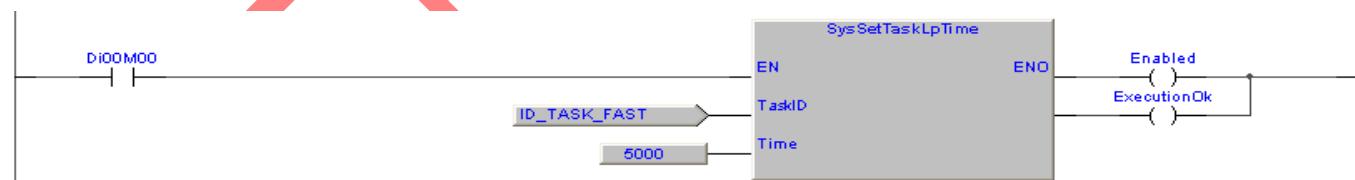
Activating the **Di00M00** input, an execution time of 5 ms is set for the task PlcFast.

Warning! To change the default execution time of tasks, you should execute the function in the boot task.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Enabled	BOOL	Auto	No	FALSE	..	Function enabled
2	ExecutionOk	BOOL	Auto	No	FALSE	..	Function execution ok

LD example (PTP116A000, LD_SysSetTaskLpTime)



ST example

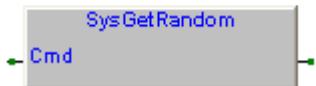
```
Enabled:=Di00M00; (* Function enabled *)
```

```
IF Di00M00 THEN
    ExecutionOk:=SysSetTaskLpTime(TaskID:=ID_TASK_FAST, Time:=5000); (* Function execution ok *)
END_IF;
```

7.5.3 SysGetRandom, get random number

Type	Library
Function	XTarget_07_0

This function returns a random number between 0.0 and 1.0. With the value of **Cmd** you can define whether you want a new random number (**Cmd:=TRUE**) or the one stored with the previous execution of the function (**Cmd:=FALSE**).



Function parameters:

Cmd (BOOL) Indicates the random number returned.

TRUE: Is saved and returned a new random number.

FALSE: The number returned is the number saved from the previous call with **Cmd:=TRUE**.

Return value:

(REAL) A random number in the range from 0.0 to 1.0.

Examples

Activating the **Di00M00** digital input, is sent on the serial port **COM0** a sequence of 10 random numbers.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	RandomNr	UINT	Auto	No	0	..	Random number
2	i	USINT	Auto	No	0	..	Auxiliary counter
3	NrOfChars	INT	Auto	No	0	..	Number of printed chars
4	Fp	FILEP	Auto	No	0	..	Port COM0 file pointer
5	Pulse	BOOL	Auto	No	FALSE	..	Pulse flag

ST example (PTP116A000, ST_SysGetRandom)

```
(* Here the COM0 port is opened in read/write, *)
IF (Fp = NULL) THEN
    Fp:=Sysfopen('COM0', 'rw'); (* Port COM0 file pointer *)
END_IF;

(* Check if input is activated. *)
F (Di00M00 <> Pulse) THEN
    Pulse:=Di00M00; (* Pulse flag *)

(* On input raising edge print out 10 random numbers. *)

    IF (Di00M00) THEN
        FOR i:=0 TO (9) BY 1 DO
            RandomNr:=TO_UINT(SysGetRandom(TRUE)*1000.0); (* Random number *)
            NrOfChars:=SysVarfprintf(Fp, 'Rn:%03d$r$\n', UINT_TYPE, ADR(RandomNr));
        END_FOR;
    END_IF;
END_IF;
```

By connecting a terminal set to **115200,8,n,1**, on serial port **COM0**, we will see a list of type:

Rn:437
Rn:488
Rn:898
...
Rn:261
Rn:944

7.5.4 SysGetLastError, get last error

Type	Library
Function	XTarget_07_0

This function returns the last error number reported by a function and/or a function block. You must execute this function on fault bit activation out from function and/or function block. Setting **Cmd**, it is possible to choice if you want to have the present value of the last error (**Cmd:=TRUE**) or the one stored with the previous execution of the function (**Cmd:=FALSE**).

Function parameters:

Cmd (BOOL) Indicates the error number returned.

TRUE: It returns the last error value.

FALSE: The number returned is the number saved from the previous call with **Cmd:=TRUE**.

Return value:

(UDINT) The number of the last error found

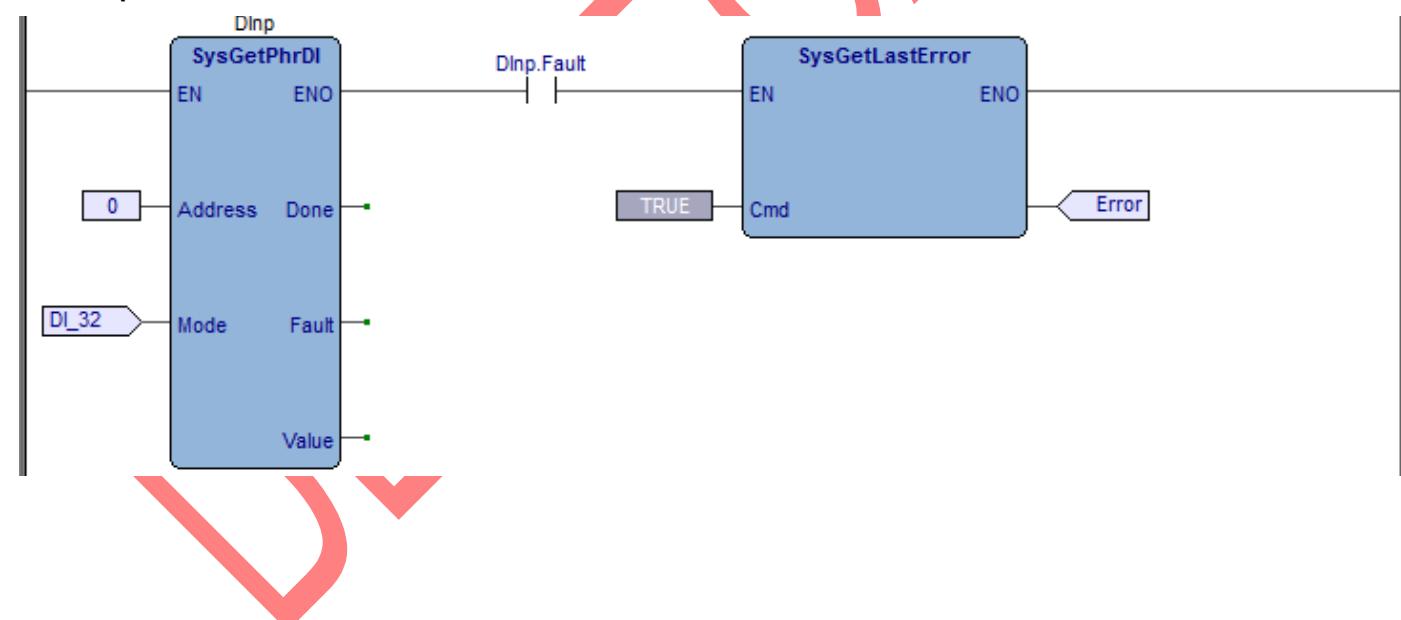
Examples

It is saved the error during the execution of the **SysGetPhrDI** function block. In case of error, the error number is transferred into the **Error** variable.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	DInp	SysGetPhrDI	Auto	No	Get peripheral inputs
2	Error	UDINT	Auto	No	Error number

LD example



7.5.5 SysPCodeAccept, accepts the protection code

Type	Library
Function	XTarget_07_0

Some features of the program and/or function blocks may be protected by a code that must be ordered separately. To enable the execution of the function and/or function block you must unlock the code with this function.

The function checks the provided code and returns **TRUE** if the code is accepted. See the [functions and function blocks protection](#) chapter for further information.

Function parameters:

Code (STRING[20]) Protection code.

Return value:

(BOOL) **TRUE**: Code ID verified its unlocked.
FALSE: Code not verified.

Error codes

If an error occurs the function returns FALSE and [SysGetLastError](#) can detect the error code.

9991100 Wrong **Code**.

9991990 Not yet implemented in the simulator.

Examples

It is showed a simple program that executes control over code "abcdefghijklmнопqrst". If the code is correct the **CodeAccepted** variable is activated.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	CodeAccepted	BOOL	Auto	No	FALSE	..	Protection code accepted
2	Enabled	BOOL	Auto	No	FALSE	..	Function enabled

LD example (PTP116A000,LD_SysPCodeAccept)



ST example

(* Check the protection code. *)
 CodeAccepted:=SysPCodeAccept('abcdefghijklmнопqrst'); (* Protection code accepted *)

7.5.6 SysCalcCrc, CRC calculation

Type	Library
Function	XTarget_12_0

Questa funzione esegue il calcolo del CRC **Cyclic Redundancy Check**, (Controllo Ciclico di Ridondanza) su di un'area dati. Il calcolo è effettuato secondo il tipo di CRC indicato nel parametro **CrcType**.

Occorre passare alla funzione l'indirizzo del buffer di memoria **Buf** ed il numero di bytes **ByteNr** su cui eseguire il calcolo del CRC. Occorre anche indicare un valore di inizializzazione del calcolo che cambia in funzione del tipo di CRC calcolato.

- Buf
- ByteNr
- CrcIni
- CrcType

Parametri funzione:

Buf (@USINT) Indirizzo dell'area di memoria su cui eseguire il calcolo del CRC.

ByteNr (UINT) Numero di bytes su cui eseguire il calcolo del CRC a partire dall'indirizzo definito in **Buf**.

CRCIni (`UINT`) Valore di inizializzazione del CRC da calcolare.

CRCType (UISNT) Tipo di CRC (1: Modbus RTU)

La funzione ritorna:

(UDINT) Valore CBC calcolato

Codici di errore

In caso di errore con **SysGetLastError** è possibile rilevare il codice di errore.

9938100 Tipo CRC errato.

Esempi

Viene calcolato il CRC di un frame modbus RtU per il comando di lettura registri ***Read holding registers***. Il valore del CRC calcolato è 17466 (16#443A).

Definizione variabili

	Name	Type	Address	Array	Init value	Attribute		Description
1	Frame	USINT	Auto	[0..9]		Frame array
2	RegsAddress	UINT	Auto	No		Registers address
3	NrOfRegs	USINT	Auto	No		Number of registers
4	CRCValue	UINT	Auto	No		CRC value

Esempio ST ($PTP116B000$, ST SysCalcCrc)

```
(* Define the registers address and the number of registers to read. *)
(* +---+---+---+---+---+---+ *)
(* |Nd|03|Addr |NumR |CRC| *)
(* +---+---+---+---+---+---+ *)

RegsAddress:=16#0120; (* Registers address *)
NrOfRegs:=8; (* Number of registers *)

(* Prepare the command frame. *)

Frame[0]:=1; (* Node address *)
Frame[1]:=3; (* Function code (16#03) *)
Frame[2]:=TO_UINT(RegsAddress/256); (* MSB registers address *)
Frame[3]:=TO_UINT(RegsAddress&255); (* LSB registers address *)
Frame[4]:=0; (* MSB number of registers to read *)
Frame[5]:=NrOfRegs; (* LSB number of registers to read *)

(* Calculate the frame CRC. *)

CRCValue:=TO_UINT(SysCalcCrc(Buf:=ADR(Frame[0]), ByteNr:=6, CrcIni:=16#FFFF, CrcType:=1)); (* CRC value *)
Frame[6]:=TO_UINT(CRCValue/256); (* MSB of CRC value *)
Frame[7]:=TO_UINT(CRCValue&255); (* LSB of CRC value *)
```

7.5.7 SysMAlloc, memory allocation

Type	Library
Function	XTarget_07_0

This function allocates a memory space of the size in bytes defined by **Size** parameter. The function returns the pointer to the allocated memory space.

The memory is allocated in system memory and then does not use the memory available to the user program. In the event that there is not room in memory for the defined buffer allocation, the function returns **NULL**.

Function parameters:

Size (UDINT) Size in bytes of the area to allocate.

Return value:

(@USINT) Address allocation buffer. **NULL** if there is no space to allocate the buffer.

Error codes

If an error occurs the function returns **NULL** and [SysGetLastError](#) can detect the error code.

9947990 Not yet implemented in the simulator.

Examples

On rising edge of **Di00M00** input, the **Counter** variable is incremented and the value is formatted on **StringOut** array. The value in **StringOut** is then sent to the serial port **COM0**.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Pulse	BOOL	Auto	No	FALSE	..	Pulse flag
2	Ch	INT	Auto	No	0	..	Written character
3	i	INT	Auto	No	0	..	Auxiliary counter
4	NrOfChars	INT	Auto	No	0	..	Number of printed chars
5	Counter	UDINT	Auto	No	0	..	Counter
6	Fp	FILEP	Auto	No	0	..	Port COM0 file pointer
7	StringOut	@USINT	Auto	No	0	..	String output pointer

ST example (PTP116A300, ST_SysMAlloc)

```
(* Here at first program execution loop allocate memory and open COM. *)

IF (SysFirstLoop) THEN
    StringOut:=SysMAlloc(16); (* String output pointer *)
    Fp:=Sysfopen('COM0', 'rw'); (* Port COM0 file pointer *)
END_IF;

IF ((StringOut = 0) OR (Fp = 0)) THEN RETURN; END_IF;

(* On input raising edge the counter value is printed. *)

IF (Di00M00 <> Pulse) THEN
    Pulse:=Di00M00; (* Pulse flag *)

    IF (Di00M00) THEN
        Counter:=Counter+1; (* Counter *)
        NrOfChars:=SysVarnprintf(StringOut, 32, 'Counter:%04d$r$\n', UDINT_TYPE, ADR(Counter));

        FOR i:=0 TO NrOfChars DO
            Ch:=Sysfputc(TO_INT(@StringOut), Fp); (* Written character *)
            StringOut:=StringOut+1; (* String output pointer *)
        END_FOR;
    END_IF;
END_IF;
```

7.5.8 SysRMAloc, relocatable memory allocation

Type	Library
Function	XTarget_12_0

Questa funzione esegue l'allocazione di uno spazio di memoria della dimensione in byte definita da parametro **Size**. L'indirizzo alla memoria allocata è salvato nella variabile **DataPtr**. La funzione ritorna **FALSE** se errore allocazione e **TRUE** se memoria allocata.



La memoria è allocata nella memoria di sistema e quindi non utilizza la memoria a disposizione del programma utente. La memoria allocata viene automaticamente rilocata dal sistema operativo per ottimizzare lo spazio. Quindi prima di utilizzarla occorre sempre fare riferimento all'indirizzo memorizzato in **DataPtr**.

Parametri funzione:

Size (UDINT) Dimensione in bytes dell'area da allocare.

DataPtr (UDINT) Buffer indirizzo memoria allocata.

La funzione ritorna:

(BOOL) **FALSE**: Errore allocazione memoria, **TRUE**: Memoria allocata

Codici di errore

In caso di errore la funzione torna **NULL** e con [SysGetLastError](#) è possibile rilevare il codice di errore.

9933100 Funzione eseguita in task fast.

9933110 Valore di **Size** errato.

9933990 Non implementata nel simulatore.

Esempi

Su fronte attivazione ingresso **Di00M00** viene incrementata la variabile **Counter** e la stampa del suo valore trasferita nell'array **StringOut**. Il valore presente in **StringOut** viene poi inviato sulla porta seriale **COM0**.

Definizione variabili

	Name	Type	Address	Array	Init value	Attribute	Description
1	Pulse	BOOL	Auto	No	..	Pulse flag	
2	NrOfChars	INT	Auto	No	..	Number of printed chars	
3	Counter	UDINT	Auto	No	..	Counter	
4	Fp	FILEP	Auto	No	..	COM port file pointer	
5	StringOut	UDINT	Auto	No	..	String output pointer	
6	Dummy	BOOL	Auto	No	..	Dummy variable	

Esempio ST (PTP116B000, ST_SysRMAloc)

```

(* Here at first program execution loop open COM. *)
IF (SysFirstLoop) THEN Fp:=Sysfopen('COM0', 'rw'); END_IF;
(* Check if input is activated. *)
IF (Di00M00 <> Pulse) THEN
  Pulse:=Di00M00; (* Pulse flag *)

IF (Di00M00) THEN
  Counter:=Counter+1; (* Counter *)

IF (SysRMAloc(16, ADR(StringOut))) THEN
  NrOfChars:=SysVarsprintf(StringOut, 14+1, 'Counter:%04d$r$n', UDINT_TYPE, ADR(Counter));
  NrOfChars:=Sysfwrite(StringOut, NrOfChars, 1, Fp);
  Dummy:=SysRMFree(ADR(StringOut));
END_IF;
END_IF;
END_IF;
  
```

7.5.9 SysRmFree, relocatable memory free

Type	Library
Function	XTarget_12

Questa funzione esegue la disallocazione di uno spazio di memoria precedentemente allocato da una funzione **SysRMAlloc**. Occorre fornire l'indirizzo alla memoria allocata nella variabile **DataPtr**. La funzione ritorna **FALSE** se errore disallocazione e **TRUE** se memoria disallocata.



Nota: Non è possibile disallocare la memoria allocata con la funzione **SysMAlloc**.

Parametri funzione:

DataPtr (UDINT) Buffer indirizzo memoria allocata.

La funzione ritorna:

(BOOL) **FALSE**: Errore disallocazione memoria, **TRUE**: Memoria disallocata

Codici di errore

In caso di errore la funzione torna **NULL** e con [SysGetLastError](#) è possibile rilevare il codice di errore.

9934100 Funzione eseguita in task fast.

9934990 Non implementata nel simulatore.

Esempi

Vedere esempio fornito con funzione **SysRMAlloc**.

DEPRECATED

7.5.10 SysGetEndianness, get the system endianness

Type	Library
Function	XTarget_12_0

Questa funzione ritorna il tipo di endianness del sistema.

Se sistema **Little-Endian**, memorizzazione che inizia dal byte meno significativo per finire col più significativo, la funzione ritorna **FALSE**.

Se sistema **Big-Endian**, memorizzazione che inizia dal byte più significativo per finire col meno significativo, la funzione ritorna **TRUE**.

Parametri funzione:

Cmd (BOOL) Deve sempre essere **TRUE**.

La funzione ritorna:

(BOOL) **FALSE**: Sistema **Little-Endian**, **TRUE**: Sistema **Big-Endian**

DEPRECATED

7.5.11 SysGetUTCDateTime, get the system Date/Time on UTC

Type	Library
Function	XTarget_12_0

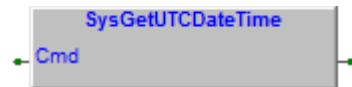
Questa funzione ritorna la Data/Ora di sistema in UTC, il valore è espresso in Epoch Time.

Parametri funzione:

Cmd (BOOL) Deve sempre essere **TRUE**.

La funzione ritorna:

(UDINT) Valore Data/Ora di sistema in UTC, il valore è espresso in Epoch Time.



DEPRECATED

7.5.12 SysSetUTCDateTime, set the system Date/Time on UTC

Type	Library
Function	XTarget_12_0

Questa funzione imposta la Data/Ora di sistema in UTC, il valore è espresso in Epoch Time.

Parametri funzione:

UTCDatetime (UDINT) Valore Data/Ora di sistema in UTC, il valore è espresso in Epoch Time.

La funzione ritorna:

(BOOL) FALSE: Errore esecuzione, TRUE: Esecuzione Ok.

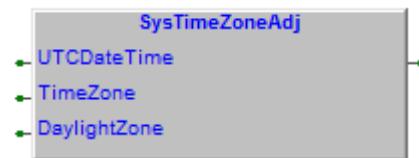
DEPRECATED

7.5.13 SysTimeZoneAdj, adjust date and time

Type	Library
Function	XTarget_12_0

Questa funzione permette di calcolare il valore di Data/Ora locale partendo dal valore UTC utilizzando il valore di **TimeZone** e **DaylightZone** definiti.

Parametri funzione:



- UTCDateTime** (UDINT) Valore di Date/Time in epoch time (UTC).
- TimeZone** (SINT) Fuso orario numero che indica la differenza in ore dell'ora locale rispetto al Tempo Coordinato Universale (UTC) riferito al meridiano di Greenwich. Per l'Italia il valore da definire è +1.
- DaylightZone** (USINT) Zona di cambio ora legale, il sistema provvede automaticamente al cambio di ora in base alla zona definita. Le zone sono 3 (Per l'Italia occorre impostare 1).
- 0) Nessun cambiamento di ora legale.
 - 1) Europa, ora legale da Aprile ad Ottobre.
 - 2) USA, ora legale da Aprile a Novembre.

La funzione ritorna:

- (UDINT) Valore di Date/Time in epoch time (Locale).

Codici di errore

In caso di errore la funzione torna **FALSE** e con [SysGetLastError](#) è possibile rilevare il codice di errore.

9940100 Errore TimeZone

9940110 Errore Daylight

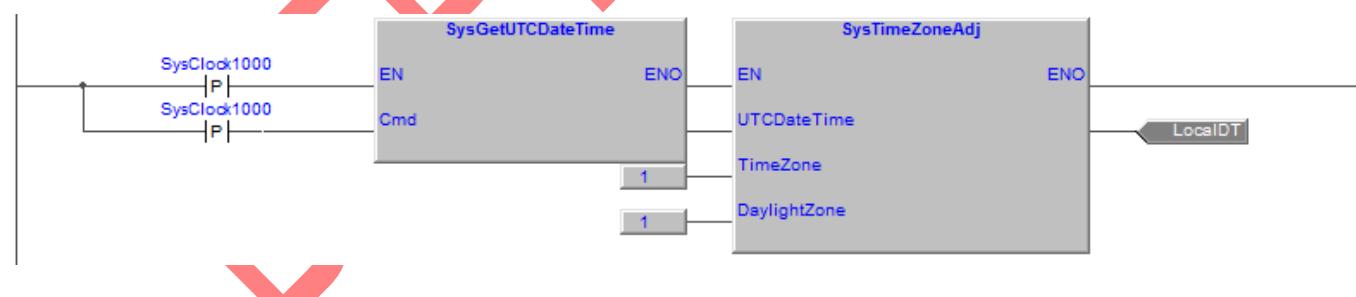
Esempi

Ad ogni secondo viene calcolato il valore di Data/Ora locale per l'Italia.

Definizione variabili

	Name	Type	Address	Array	Init value	Attribute	Description
1	LocalDT	UDINT	Auto	No	..		Local Date/Time

Esempio LD ([PTP116B000](#), [LD_SysTimeZoneAdj](#))



7.5.14 SysSpyData, system spy data

Type	Library
Function	XTarget_11_0

This function allows to send data to the spy console (Accessible by Telnet with **SpyData** command). You can define the way how the spy data is displayed Mode and a **Label** returned in the spy string.

The **TFlags** parameter defines a 32 bits pattern which is used as a trigger for the data display in the Telnet spy console.

By executing the function with all parameters to "0" is checked the space in the Telnet spy console. The function returns TRUE if there is space to save the spy record and FALSE if the console is busy.



Function parameters:

- | | |
|-----------------------|---|
| Mode (USINT) | Defines the way how the spy data is displayed, Spy mode . |
| TFlags (UDINT) | Defines a pattern which is used as a trigger for the data display. |
| Label (@USINT) | Label returned in the spy string. |
| DPtr (@USINT) | Pointer to data to spy |

La funzione ritorna:

(BOOL) **TRUE**: Spy available. **FALSE**: Spy console busy.

Error codes

If an error occurs, with [SysGetLastError](#) you can detect the error code.

9950100 Function executed in task fast or slow.

9950200 Memory allocation error.

9950990 Not yet implemented in the simulator.

Spy record

The spy record appears in the spy console with a total length of 80 characters, typically is a record of the type:

00:00:00(0000)|Label|Spy data string-----

The time and the delay by the previous spy records fields have a constant length (15 characters). **Field Label** and spy data can reach a total of 65 characters.

Example

This sample program sends to each second three records to the spy console.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	TString	STRING	Auto	[32]		..	Text string
2	SpyPulse	BOOL	Auto	No	FALSE	..	Spy data pulse
3	ABf	BOOL	Auto	No	FALSE	..	Auxiliary buffer

ST examples (PTP116A400, ST_SysSpyData)

```
(* Init the string to be spied and check if there is space to spy. *)
TString:='Hello!$r$\n'; (* Text string *)
IF NOT(SysSpyData(0, 0, NULL, NULL))THEN RETURN; END_IF;

(* Check if is a second pulse. *)

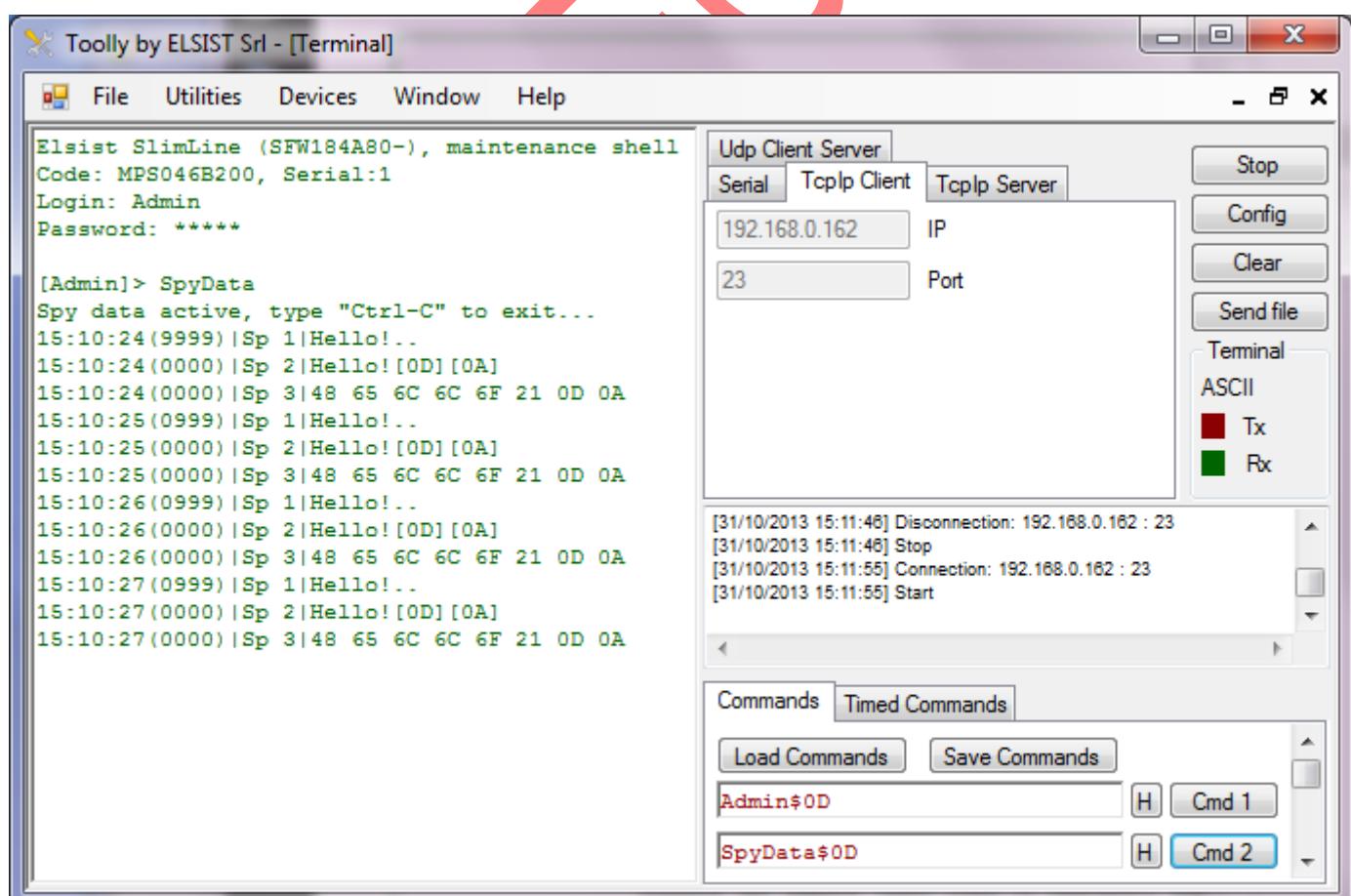
IF (SysClock1000 = SpyPulse) THEN RETURN; END_IF;
SpyPulse:=SysClock1000; (* Spy data pulse *)

(* Send 3 spy data records. *)

ABF:=SysSpyData(SPY_ASCII, 16#00000001, ADR('Sp 1'), ADR(TString));
ABF:=SysSpyData(SPY_ASCHEX, 16#00000010, ADR('Sp 2'), ADR(TString));
ABF:=SysSpyData(SPY_BINARY+8, 16#00000100, ADR('Sp 3'), ADR(TString));
```

Spy console

To activate the spy console, you must activate a Telnet connection to the system, (Refer to the SlimLine CPU's Telnet commands reference manual). With the **SpyData** command you activate the spy console and displays the various data records.



7.6 Functions and FBs for Date/Time management

7.6.1 SysETimeToDate, epoch time to date conversion

Type	Library
FB	XTarget_07_0

This function block performs the conversion of data expressed in epoch time. It should provide the epoch time value as present in the [SysDateTime](#) system variable. Output from the function block will have date values expressed in the format Day/Month/Year and Time:Minutes:Seconds.



EpochTime (UDINT) You must specify the date expressed in epoch time.

Done (BOOL) Activated at the end of conversion.

Fault (BOOL) Conversion error. Is activated in case of error in the conversion.

Year (UINT) Return the year value (Range from 1970 to 2099)

Month (USINT) Return the month of year (Range from 1 to 12)

Day (USINT) Return the day of month (Range from 1 to 31)

WeekDay (USINT) Return the weekday value (Range from 0 to 6)
0: Sunday, 1:Monday, 2:Tuesday, 3:Wednesday, 4:Thursday, 5:Friday, 6:Saturday

Hour (USINT) Return the hour value (Range from 0 to 23)

Minute (USINT) Return the minute value (Range from 0 to 59)

Second (USINT) Return the seconds value (Range from 0 to 59)

Error codes

If an error occurs the **Fault** output is activated and [SysGetLastError](#) can detect the error code.

9986050 Function block allocation error.

9986060 Function block version error.

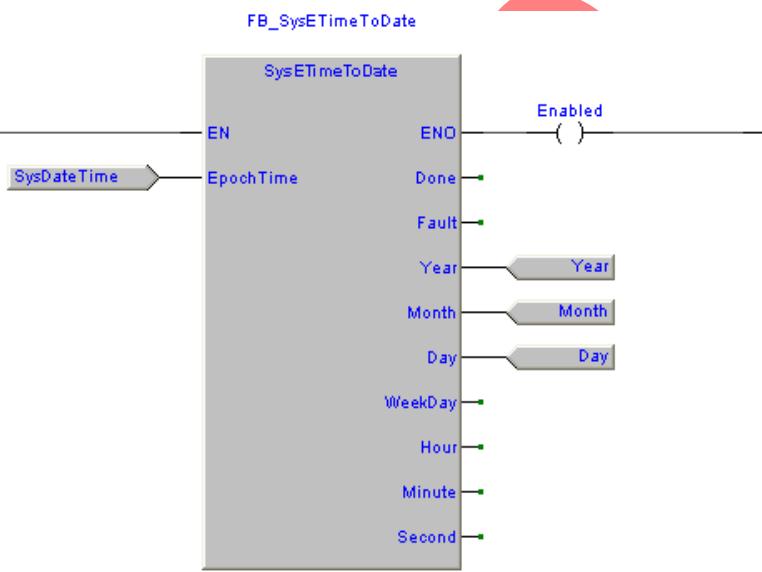
Examples

It is converted the epoch time value from **SysDateTime** variable to year, month and day variables.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	FB_SysETimeToDate	SysETimeToDate	Auto	No	0	..	FB SysETimeToDate data
2	Year	UINT	Auto	No	0	..	Year
3	Month	USINT	Auto	No	0	..	Month
4	Day	USINT	Auto	No	0	..	Day

LD example (PTP116A100, LD_SysETimeToDate)



IL example (PTP116A100, IL_SysETimeToDate)

```
(* Transfer system date e time to FB input variable. *)
LD SysDateTime
ST FB_SysETimeToDate.EpochTime

CAL FB_SysETimeToDate (* Call the SysETimeToDate function block *)

(* Transfer the FB output variables to program variables. *)
LD FB_SysETimeToDate.Year
ST Year

LD FB_SysETimeToDate.Month
ST Month

LD FB_SysETimeToDate.Day
ST Day
```

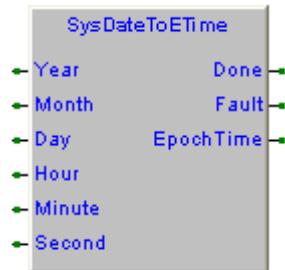
ST example (PTP116A100, ST_SysETimeToDate)

```
(* Here FB SysETimeToDate is executed and variables copied. *)
FB_SysETimeToDate.EpochTime:=SysDateTime;
FB_SysETimeToDate();
Year:=FB_SysETimeToDate.Year; (* Year *)
Month:=FB_SysETimeToDate.Month; (* Month *)
Day:=FB_SysETimeToDate.Day; (* Day *)
```

7.6.2 SysDateToETime, date to epoch time conversion

Type	Library
FB	XTarget_07_0

This function block performs the conversion of date-time in epoch time. You must provide the date and time input and in the function block output, you will have a value in epoch time.



Year (UINT)	Define the year value (Range from 1970 to 2099).
Month (USINT)	Define the month of the year (Range from 1 to 12).
Day (USINT)	Define the day of month (Range from 1 to 31).
Hour (USINT)	Define the hour value (Range from 0 to 23).
Minute (USINT)	Define the minute value (Range from 0 to 59).
Second (USINT)	Define the second value (Range from 0 to 59).
Done (BOOL)	Activated at the end of conversion.
Fault (BOOL)	Conversion error. Is activated in case of error in the conversion.
EpochTime (UDINT)	Converted epoch time value

Error codes

If an error occurs the **Fault** output is activated and [SysGetLastError](#) can detect the error code.

- 9987050 Function block allocation error.
- 9987060 Function block version error.
- 9987200 Error during function block execution.

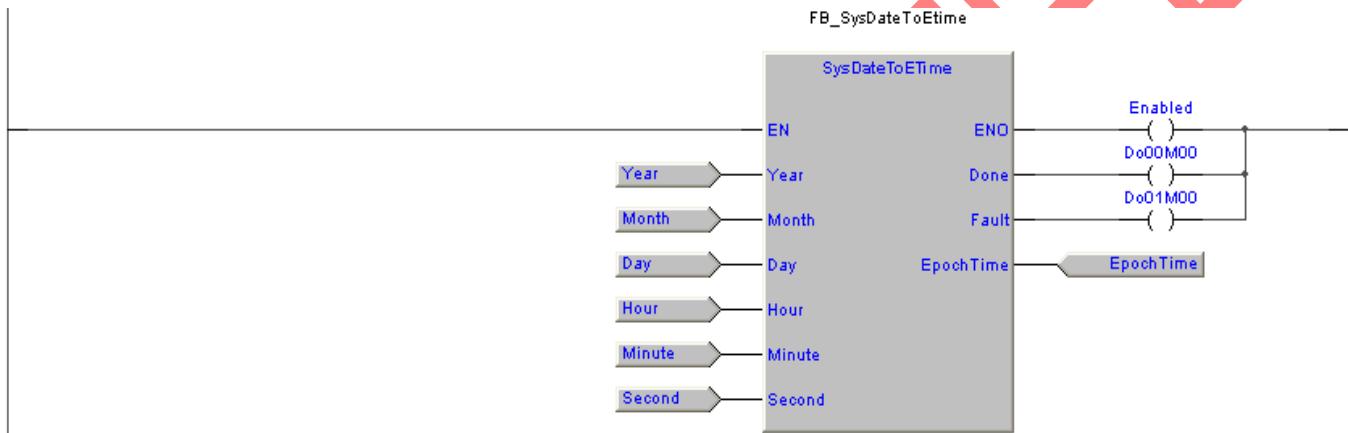
Examples

The date-time value is converted in epoch time. Defining the value of such date and time 9/4/2010 14:20:15 will output the value 1270822815.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	FB_SysDateToEtime	SysDateToETime	Auto	No	0	..	FB SysDateToETime data
2	Hour	USINT	Auto	No	0	..	Day
3	Minute	USINT	Auto	No	0	..	Month
4	Second	USINT	Auto	No	0	..	Year
5	Day	USINT	Auto	No	0	..	Day
6	Month	USINT	Auto	No	0	..	Month
7	Year	UINT	Auto	No	0	..	Year
8	EpochTime	UDINT	Auto	No	0	..	Epoch time

LD example (PTP116A100, LD_SysDateToETime)



IL example (PTP116A100, IL_SysDateToETime)

```

(* Transfer date e time to FB input variable. *)
LD Year
ST FB_SysDateToEtime.Year
LD Month
ST FB_SysDateToEtime.Month
LD Day
ST FB_SysDateToEtime.Day
CAL FB_SysDateToEtime (* Call the SysDateToEtime function block *)
(* Transfer the FB output variables to program variables. *)
LD FB_SysDateToEtime.EpochTime
ST EpochTime

```

ST example (PTP116A100, ST_SysDateToETime)

```

(* Here FB SysDateToETime is executed and variables copied. *)
FB_SysDateToEtime.Year:=Year;
FB_SysDateToEtime.Month:=Month;
FB_SysDateToEtime.Day:=Day;
FB_SysDateToEtime(); (* Call the SysDateToEtime function block *)
EpochTime:=FB_SysDateToEtime.EpochTime; (* Epoch time *)

```

7.7 Functions and FBs for I/O terminal management

7.7.1 Sysfopen, file open

Type	Library
Function	XTarget_07_0

This function allows the opening of a resource specified by the **FName** parameter, linking it with a **stream** of data usable in subsequent calls to the I/O functions. The function returns the pointer to the resource.

If the resource specified is already open or the name of the resource is incorrect, the function returns **NULL**. If you are opening a disk file to create it, make sure the disk is formatted.



Function parameter:

FName (STRING[20])

Name of the resource to use.

Name	Resource
COM0	Serial port COM0
COM1	Serial port COM1
COM2	Serial port COM2
PCOMx.y	Serial port y on extension module with x address
UDPSKT	UDP socket
TCPSKT	TCP socket
pathname	Full path with filename also (example.: 'Storage/myFile.txt')

Mode (STRING[4])

Indicates how the resource is open: r=read, w=write a=append. For serial ports, use 'rw'. To create a file on disk, you should try opening with 'w' or 'a'. The opening with 'w' on an existing file, erase its contents. The opening with 'r' or 'w' position the position indicator of the stream at the beginning of the file. Opening with 'a' positions it at the end.

Return value:

(FILEP)

Pointer to resource.

NULL: In case of error.

Error codes

If an error occurs the function returns with **NULL** and [SysGetLastError](#) can detect the error code.

9996100 The **FName** resource has wrong length.

9996110 The **FName** resource has wrong length.

9996200~2 Impossible to use port from user program.

9996990 Not yet Implemented in the simulator.

Example

The serial port is open in read/write mode.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Fp	FILEP	Auto	No	0	..	Port COM0 file pointer

LD example



ST example (PTP116A100, ST_Sysfopen)

```

IF (Fp = NULL) THEN
    Fp:=Sysfopen('COM0', 'rw'); (* Port COM0 file pointer *)
END_IF;
    
```

DEPRECATE

7.7.2 SysFIsOpen, get the file open status

Type	Library
Function	XTarget_12_0

Questa funzione controlla se File è aperto.

La funzione ritorna TRUE se file aperto.

Parametri funzione:

File (FILEP) Flusso dati **stream** ritornato dalla funzione **Sysfopen**.

La funzione ritorna:

(BOOL) **FALSE**: File non corretto o chiuso. **TRUE**: File aperto.

Codici di errore

In caso di errore la funzione torna con **EOF** e con **SysGetLastError** è possibile rilevare il codice di errore.

9973990 Non implementata nel simulatore.

Esempi

Viene aperta e successivamente chiusa la porta seriale **COM0**. Se la porta è correttamente chiusa viene attivata l'uscita **Do01M00**.

Definizione variabili

	Name	Type	Address	Array	Init value	Attribute	Description
1	Fp	FILEP	Auto	No	0	..	Port COM0 file pointer

Esempio ST (PTP116B000, ST_SysFIsOpen)

```
(* Here the COM0 port is opened in read/write. *)
IF (Fp = NULL) THEN Fp:=Sysfopen('COM0', 'rw'); END_IF;
(* Here the COM0 port is closed. *)
IF (SysFIsOpen(Fp)) THEN
  Do01M00:=(Sysfclose(Fp) <> EOF); (* Output is set if port is closed *)
END_IF;
```

7.7.3 Sysclose, file close

Type	Library
Function	XTarget_07_0

This function allows closing the connection to the resource indicated by the **File** parameter previously opened by the [Sysopen](#) function.

If case of error, the function returns **EOF**.

Function parameters:

File (FILEP) **Stream** returned from [Sysopen](#) function.

Return value:

(INT) **0**: If execution ok.
EOF: In case of error.

Error codes

If an error occurs the function returns with **EOF** and [SysGetLastError](#) can detect the error code.

9973100 I/O terminal used in task fast or slow.

9973200 Error closing resource.

9973990 Not yet Implemented in the simulator.

Examples

The serial port **COM0** is opened and then closed. If the port is open **Do00M00** output is activated. If the port is correctly closed **Do01M00** output is activated. Viene aperta e successivamente chiusa la porta seriale **COM0**.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Fp	FILEP	Auto	No	0	..	Port COM0 file pointer

ST example (PTP116A100, ST_Sysclose)

```
(* Here the COM0 port is opened in read/write. *)
IF (Fp = NULL) THEN
    Fp:=Sysopen('COM0', 'rw'); (* Port COM0 file pointer *)
    Do00M00:=(Fp <> NULL); (* Output is set if port is opened *)
END_IF;

(* Here the COM0 port is closed. *)
IF (Fp <> NULL) THEN
    Do01M00:=(Sysclose(Fp) <> EOF); (* Output is set if port is closed *)
END_IF;
```

7.7.4 Sysfgetc, get character from file

Type	Library
Function	XTarget_07_0

This function returns a character from the **stream** indicated by the **File** parameter previously opened by the [Sysfopen](#) function.

The function returns the character read from the stream. If an error occurs or if no data available from the stream, the function returns **EOF**. To make sure that there are characters from the stream you can use the [SysGetIChars](#) function that returns they number.

Function parameters:

File (FILEP) Stream returned by [Sysfopen](#) function.

Return value:

(INT) Read character from stream.

EOF: In case of error or no data available from stream.

Error codes

If an error occurs the function returns with **EOF** and [SysGetLastError](#) can detect the error code.

9972100 I/O terminal used in task fast or slow.

9972990 Not yet Implemented in the simulator.

Examples

Serial port **COM0** is opened and checked if there are characters available. If at least one character is available it is read and transferred into the **Ch**.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Fp	FILEP	Auto	No	0	..	Port COM0 file pointer
2	Ch	INT	Auto	No	0	..	Character read

ST example (PTP116A100, ST_Sysfgetc)

```
(* Here the COM0 port is opened in read/write. *)
IF (Fp = NULL) THEN
    Fp:=Sysfopen('COM0', 'rw'); (* Port COM0 file pointer *)
END_IF;

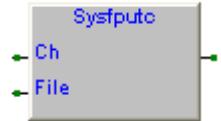
(* Here check if a character is available from port and read it. *)
IF (Fp <> NULL) THEN
    IF (TO_BOOL(SysGetIChars(Fp))) THEN
        Ch:=Sysfgetc(Fp); (* Get input character *)
    END_IF;
END_IF;
```

7.7.5 Sysputc, put character to file

Type	Library
Function	XTarget_07_0

This function send a character into the **stream** indicated by the **File** parameter previously opened by the [Sysfopen](#) function.

The function returns the character written to the stream. In case of error or if the stream does not accept the data, the function returns EOF. To ensure that there is room to accept the character to the stream, you can use the [SysGetOSpace](#) function that returns the available space.



Function parameter:

Ch (INT) Character to send to stream.

File (FILEP) Stream returned by [Sysfopen](#) function.

Return value:

(INT) Character written to stream. **EOF**: In case of error or no data available from stream. In caso di errore o se lo stream non accetta il dato.

Error codes

If an error occurs the function returns with **EOF** and [SysGetLastError](#) can detect the error code.

9971100 I/O terminal used in task fast or slow.

9971990 Not yet Implemented in the simulator.

Examples

It is showed a simple program that performs the echo of characters received from the serial port **COM0**. The **COM0** serial port is opened and checked and if there are characters available. If at least one character is available it is read and subsequent written.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Fp	FILEP	Auto	No	0	..	Port COM0 file pointer
2	Ch	INT	Auto	No	0	..	Character read

ST examples (PTP116A100, ST_Sysputc)

```

(* Here the COM0 port is opened in read/write. *)

IF (Fp = NULL) THEN
  Fp:=Sysfopen('COM0', 'rw'); (* Port COM0 file pointer *)
END_IF;

(* Here execute the received characters echo. *)

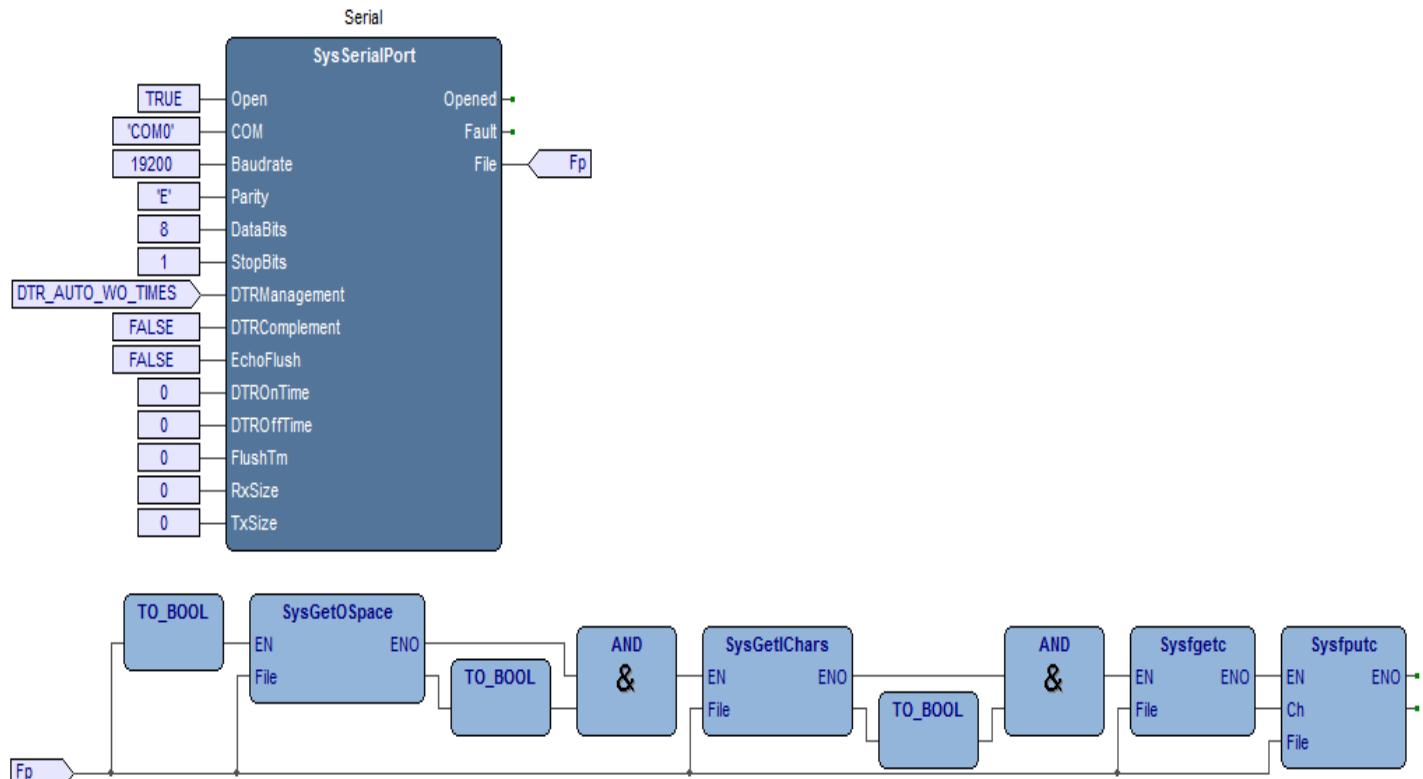
IF (Fp <> NULL) THEN
  IF (TO_BOOL(SysGetIChars(Fp))) AND (TO_BOOL(SysGetOSpace(Fp))) THEN
    Ch:=Sysfgetc(Fp); (* Get input character *)
    Ch:=Sysputc(Ch, Fp); (* Put input character *)
  END_IF;
END_IF;
  
```

Using the I/O terminal functions, it is possible to create a simple program that performs the echo of character received from the serial port **COM0**. The port is opened at 19200, e, 8, 1. All characters received are echoed.

Definizione variabili

	Name	Type	Address	Array	Init value	Attribute	Description
1	Fp	FILEP	Auto	No	File pointer array
2	Serial	SysSerialPort	Auto	No	

Esempio FBD (PTP119B000, FBD_SerialEcho)



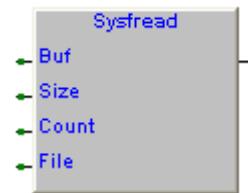
DEP

7.7.6 Sysread, read data from file

Type	Library
Function	XTarget_07_0

This function reads a specified number of strings of defined length, from the stream indicated by the **File** parameter previously opened by the [Sysfopen](#) function.

The function returns the number of read data strings. If in the **stream** there are not enough string to meet the parameters, it returns fewer string than the defined value.



Function parameter:

Buf (@STRING) Address of string where to store the read strings.

Size (INT) Length in character of the strings to read.

Count (INT) Number of strings to read.

File (FILEP) Stream returned by [Sysfopen](#) function.

Return value:

(INT) Number of strings read. If the return value is less than **Count**, mean that there not were enough data in the stream.

Error codes

If an error occurs the function returns **0** and [SysGetLastError](#) can detect the error code.

9970100 I/O terminal used in task fast or slow.

9970990 Not yet Implemented in the simulator.

Examples

At least 5 characters are waiting from the serial port and when received, a string of 5 characters (5 strings of 1 character) is read. The string read is transferred into the variable **RxString** and then sent to the serial port. Note that also in transmission a string of 5 characters (1 string of 5 characters) is sent.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Fp	FILEP	Auto	No	0	..	Port COM0 file pointer
2	RxString	STRING	Auto	[10]	0	..	Received string
3	RxChars	INT	Auto	No	0	..	Received characters
4	TxChars	INT	Auto	No	0	..	Transmitted characters

ST example (PTP116A100, ST_Sysread)

```

(* Here the COM0 port is opened in read/write. *)

IF (Fp = NULL) THEN
  Fp:=Sysfopen('COM0', 'rw'); (* Port COM0 file pointer *)
END_IF;

(* Here wait until at least 5 chars are received and echoes them. *)

IF (Fp <> NULL) THEN
  IF (SysGetIChars(Fp) >= 5) THEN
    RxChars:=Sysread(ADR(RxString), 1, 5, Fp); (* Received characters *)
    TxChars:=Sysread(ADR(RxString), 5, 1, Fp); (* Received characters *)
  END_IF;
END_IF;
  
```

7.7.7 Sysfwrite, write data to file

Type	Library
Function	XTarget_07_0

This function write a specified number of strings of defined length, to the **stream** indicated by the **File** parameter previously opened by the [Sysfopen](#) function.

The function returns the number of written data strings. If in the **stream** there are not enough space to store strings defined by parameters, it returns fewer string number than the defined value.



Function parameter:

- Buf** (@STRING) Address of string to write.
- Size** (INT) Length in character of the strings to write.
- Count** (INT) Number of strings to write.
- File** (FILEP) Stream returned by [Sysfopen](#) function.

La funzione ritorna:

- (INT) Number of strings written. If the return value is less than **Count**, there was not enough space in the stream.

Error codes

If an error occurs the function returns **0** and [SysGetLastError](#) can detect the error code.

9969100 I/O terminal used in task fast or slow.

9969990 Not yet Implemented in the simulator.

Examples

At least 5 characters are waiting from the serial port and when received, a string of 5 characters (5 strings of 1 character) is read. The string read is transferred into the variable **RxString** and then sent to the serial port. Note that also in transmission a string of 5 characters (1 string of 5 characters) is sent.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Fp	FILEP	Auto	No	0	..	Port COM0 file pointer
2	RxString	STRING	Auto	[10]	0	..	Received string
3	RxChars	INT	Auto	No	0	..	Received characters
4	TxChars	INT	Auto	No	0	..	Transmitted characters

ST example

```

(* Here the COM0 port is opened in read/write. *)

IF (Fp = NULL) THEN
  Fp:=Sysfopen('COM0', 'rw'); (* Port COM0 file pointer *)
END_IF;

(* Here wait until at least 5 chars are received and echoes them. *)

IF (Fp <> NULL) THEN
  IF (SysGetIChars(Fp) >= 5) THEN
    RxChars:=Sysfread(ADR(RxString), 1, 5, Fp); (* Received characters *)
    TxChars:=Sysfwrite(ADR(RxString), 5, 1, Fp); (* Transmitted characters *)
  END_IF;
END_IF;
  
```

7.7.8 SysGetIChars, get input available characters from file

Type	Library
Function	XTarget_07_0

Alias of SysGetIChars function.

This function returns the number of characters available for read from the **stream** indicated by the **File** parameter previously opened by the [Sysfopen](#) function.

If the return value is not **0**, the characters can be read with the function [Sysfgetc](#).

Function parameter:

File (FILEP) Stream returned by [Sysfopen](#) function.

Return value:

(INT) Number of available characters on stream.

Error codes

If an error occurs the function returns **0** and [SysGetLastError](#) can detect the error code.

9968100 I/O terminal used in task fast or slow.

9968990 Not yet Implemented in the simulator.

Examples

It is showed a simple program that performs the echo of characters received from the serial port **COM0**. The **COM0** serial port is opened and checked if there are available character. If at least one character is available, it is read and subsequent retransmitted.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Fp	FILEP	Auto	No	0	..	Port COM0 file pointer
2	Ch	INT	Auto	No	0	..	Character read

ST example

```
(* Here the COM0 port is opened in read/write. *)
IF (Fp = NULL) THEN
    Fp:=Sysfopen('COM0', 'rw'); (* Port COM0 file pointer *)
END_IF;

(* Here execute the received characters echo. *)
IF (Fp <> NULL) THEN
    IF (TO_BOOL(SysGetIChars(Fp))) AND (TO_BOOL(SysGetOSpace(Fp))) THEN
        Ch:=Sysfgetc(Fp); (* Get input character *)
        Ch:=Sysfputc(Ch, Fp); (* Put input character *)
    END_IF;
END_IF;
```

7.7.9 SysFGetOSpace, get output available space on file

Type	Library
Function	XTarget_07_0

Alias of SysGetOSpace function.

This function returns the available space for write to **stream** indicated by the **File** parameter previously opened by the [Sysfopen](#) function.

If the return value is not **0**, the characters can be written with the function [Sysfputc](#) function.

Function parameter:

File (FILEP) Stream returned by [Sysfopen](#) function.

Return value:

(INT) Available space to write to stream.
In buffer empty, the buffer dimension is returned.

Error codes

If an error occurs the function returns **0** and [SysGetLastError](#) can detect the error code.

9967100 I/O terminal used in task fast or slow.

9967990 Not yet Implemented in the simulator.

Examples

It is showed a simple program that performs the echo of characters received from the serial port **COM0**. The **COM0** serial port is opened and checked if there are available character. If at least one character is available, it is read and subsequent retransmitted.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Fp	FILEP	Auto	No	0	..	Port COM0 file pointer
2	Ch	INT	Auto	No	0	..	Character read

ST example

```
(* Here the COM0 port is opened in read/write. *)
IF (Fp = NULL) THEN
    Fp:=Sysfopen('COM0', 'rw'); (* Port COM0 file pointer *)
END_IF;

(* Here execute the received characters echo. *)
IF (Fp <> NULL) THEN
    IF (TO_BOOL(SysGetIChars(Fp))) AND (TO_BOOL(SysGetOSpace(Fp))) THEN
        Ch:=Sysfgetc(Fp); (* Get input character *)
        Ch:=Sysfputc(Ch, Fp); (* Put input character *)
    END_IF;
END_IF;
```

7.7.10 SysFGetIBfSize, get file Rx input buffer size

Type	Library
Function	XTarget_07_0

Alias of SysGetRxBSize function.

This function returns the size of the input buffer (receive) of data **stream** indicated by the **File** parameter and previously opened by the **Sysopen** function.



Function parameters:

File (FILEP) Stream returned by [Sysopen](#) function.

Return value:

(UDINT) Input buffer dimension in number of characters (Bytes).

Error codes

If an error occurs the function returns **0** and **SysGetLastError** can detect the error code.

9966100 I/O terminal used in task fast or slow.

9966990 Not yet Implemented in the simulator.

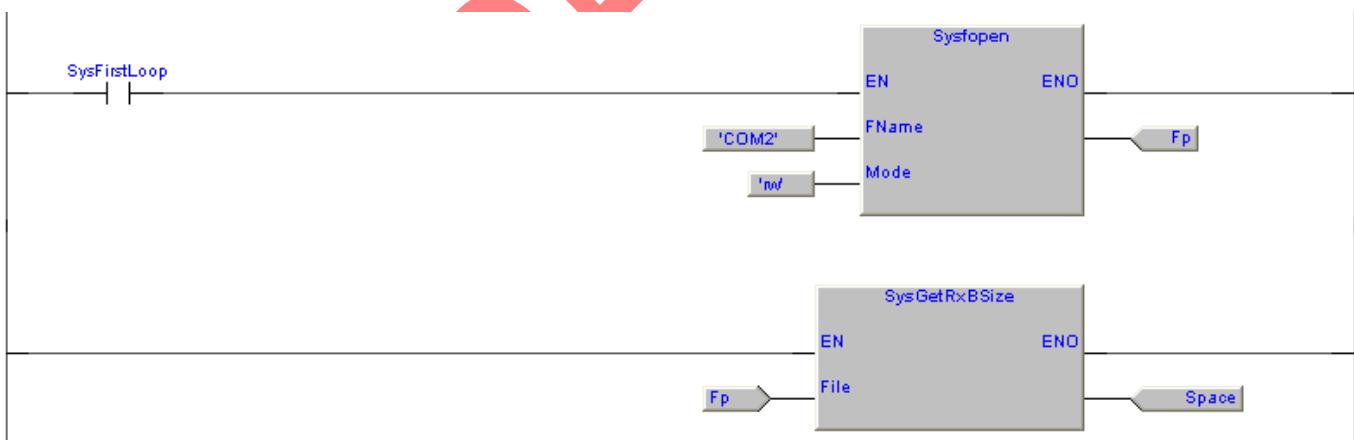
Examples

It is showed a simple program that returns the size of the input buffer (receive) of **COM2** serial port. The returned value in number of characters (bytes) is transferred to the **Space** variable.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Fp	FILEP	Auto	No	0	..	File pointer
2	Space	UDINT	Auto	No	0	..	Rx buffer space (Nr of Chars)

LD example



7.7.11 SysFGetOBfSize, get file Tx output buffer size

Type	Library
Function	XTarget_12_0

Alias of *SysGetTxBSIZE* function.

This function returns the size of the output buffer (transmit) of data **stream** indicated by the **File** parameter and previously opened by the **Sysfopen** function.



Function parameters:

File (FILEP) Stream returned by **Sysfopen** function.

Return value:

(UDINT) Output buffer dimension in number of characters (Bytes).

Error codes

If an error occurs the function returns **0** and **SysGetLastError** can detect the error code.

9965100 I/O terminal used in task fast or slow.

9965990 Not yet Implemented in the simulator.

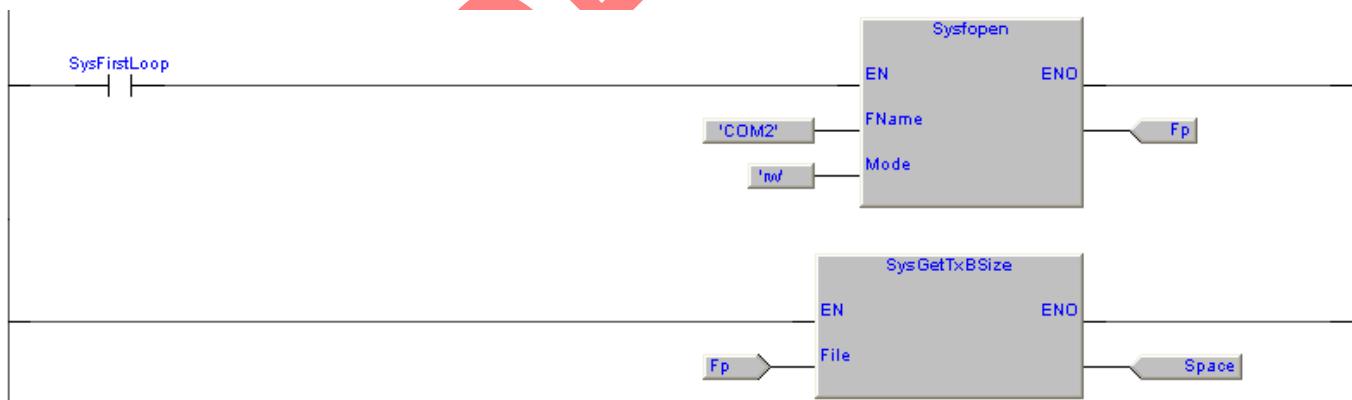
Examples

It is showed a simple program that returns the size of the output buffer (transmit) of **COM2** serial port. The returned value in number of characters (bytes) is transferred to the **Space** variable.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Fp	FILEP	Auto	No	0	..	File pointer
2	Space	UDINT	Auto	No	0	..	Rx buffer space (Nr of Chars)

LD example



7.7.12 SysFIBfClear, file input buffer clear

Type	Library
Function	XTarget_07_0

This function deletes all the input characters in the **stream** indicated by the **File** parameter previously opened by the **Sysfopen** function.

If an error occurs the function returns **FALSE**.



Function parameters:

File (FILEP) Stream returned by **Sysfopen** function.

Return value:

(BOOL) **FALSE**: Execution error.
TRUE: Execution ok.

Error codes

If an error occurs the function returns **FALSE** and **SysGetLastError** can detect the error code.

9964100 I/O terminal used in task fast or slow.

9964990 Not yet Implemented in the simulator.

Examples

If **Di00M00** input is active, all characters from the serial port input will be deleted and **Do00M00** activated.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Fp	FILEP	Auto	No	0	..	Port COM0 file pointer

ST example

```

(* Here the COM0 port is opened in read/write. *)
IF (Fp = NULL) THEN
    Fp:=Sysfopen('COM0', 'rw'); (* Port COM0 file pointer *)
END_IF;

(* If the input is active the input buffer is cleared. *)
IF (Fp <> NULL) THEN
    IF (Di00M00) THEN Do00M00:=SysFIBfClear(Fp); END_IF;
END_IF;

```

7.7.13 SysFOBfClear, file output buffer clear

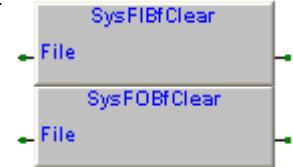
Type	Library
Function	XTarget_07_0

This function deletes all the output characters in the **stream** indicated by the **File** parameter previously opened by the **Sysfopen** function.

If an error occurs the function returns **FALSE**.

Function parameters:

File Stream returned by **Sysfopen** function.
(FILEP)



Return value:

(BOOL) **FALSE**: Execution error.
TRUE: Execution ok.

Error codes

If an error occurs the function returns **FALSE** and **SysGetLastError** can detect the error code.

9963100 I/O terminal used in task fast or slow.

9963990 Not yet Implemented in the simulator.

Examples

If **Di00M00** input is active, all characters from the serial port output will be deleted and **Do00M00** activated.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Fp	FILEP	Auto	No	0	..	Port COM0 file pointer

ST example

```

(* Here the COM0 port is opened in read/write. *)
IF (Fp = NULL) THEN
  Fp:=Sysfopen('COM0', 'rw'); (* Port COM0 file pointer *)
END_IF;

(* If the input is active the ouput buffer is cleared. *)
IF (Fp <> NULL) THEN
  IF (Di00M00) THEN Do00M00:=SysFOBfClear(Fp); END_IF;
END_IF;
  
```

7.7.14 SysFOBfFlush, file output buffer flush

Type	Library
Function	XTarget_07_0

This function forces an immediate exit of the characters present in the output **stream** indicated by the **File** parameter previously opened by the **Sysopen** function.

If an error occurs the function returns **FALSE**.



Function parameters:

File (FILEP) Stream returned by **Sysopen** function.

Return value:

(BOOL) **FALSE**: Execution error.
TRUE: Execution ok.

Error codes

If an error occurs the function returns **FALSE** and **SysGetLastError** can detect the error code.

9962100 I/O terminal used in task fast or slow.

9962990 Not yet Implemented in the simulator.

Examples

If **Di00M00** input is active, all the characters in the output buffer of the serial port will be transmitted and the **Do00M00** output activated.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Fp	FILEP	Auto	No	0	..	Port COM0 file pointer

ST example

```

(* Here the COM0 port is opened in read/write. *)
IF (Fp = NULL) THEN
    Fp:=Sysopen('COM0', 'rw'); (* Port COM0 file pointer *)
END_IF;

(* If the input is active the ouput buffer is cleared. *)

IF (Fp <> NULL) THEN
    IF (Di00M00) THEN Do00M00:=SysFOBfFlush(Fp); END_IF;
END_IF;
  
```

7.7.15 SysVarprintf, variable print to file

Type	Library
Function	XTarget_07_0

This function prints a formatted variable on the **File** stream previously opened by the **Sysfopen** function.

The **Format** string specifies the format to use to print the variable. In **VarType** there is the variable type and in **VarAdd** there is the variable address.

The function return the number of characters written into the stream. **EOF** in case of error.

Function parameters:

File (FILEP) Stream returned by **Sysfopen** function.

Format (STRING[80]) It has two types of argument: ordinary characters that are copied into the output stream and conversion specifications, denoted by the percent symbol (%) and a character that specifies the format in which to print the defined variable.

VarType (USINT) Variable type, as indicated in the [variable types definition](#) table.

VarAdd (UDINT) Variable address.

Return value:

(INT) Number of characters written into the stream. **EOF** in case of error.

Error codes

If an error occurs the function returns **EOF** and [**SysGetLastError**](#) can detect the error code.

9998010 **File** value not defined.

9968100 I/O terminal used in task fast or slow.

9998200 Variable type not managed. Check **VarType**.

9982990 Not yet Implemented in the simulator.

Examples

On rising edge of **Di00M00** input, the **Counter** variable is increased and its value is sent to **COM0** serial port. In the **NrOfChars** variable, the number of sent characters to serial port is loaded.

Defining variables

	Name	Type	Address	Array	Initvalue	Attribute	Description
1	Fp	FILEP	Auto	No	0	..	Port COM0 file pointer
2	Pulse	BOOL	Auto	No	FALSE	..	Pulse flag
3	Counter	UDINT	Auto	No	0	..	Counter
4	NrOfChars	INT	Auto	No	0	..	

ST example (PTP116A300, ST_SysVarprintf)

```
(* Here the COM0 port is opened in read/write. *)

IF (Fp = NULL) THEN Fp:=Sysfopen('COM0', 'rw'); END_IF;

IF (Fp <> NULL) THEN

    IF (Di00M00 <> Pulse) THEN
        Pulse:=Di00M00; (* Pulse flag *)

        IF (Di00M00) THEN
            Counter:=Counter+1; (* Counter *)
            NrOfChars:=SysVarprintf(Fp, 'Counter:%04d$r$\n', UDINT_TYPE, ADR(Counter));
        END_IF;
    END_IF;
END_IF;
```

7.8 File system

The "SlimLine ARM7 CPU" from firmware version **SFW167C100**, can manage the file system. In these CPUs there are two default directory:

Storage: Directory allocated on the EEPROM embedded chip in SlimLine (All versions).

SDCard: Directory allocated on SD Card to be inserted into the connector.

For formatting the file system, please refer to the user manual. The file system is accessible by FTP, then using an FTP client you can create new files, delete existing files, rename existing files, read and write data in files.

DEPRECATED

7.8.1 Sysremove, file remove

Type	Library
Function	XTarget_07_0

This function performs the removal (deletion) of a file. **Name** must be defined in the file name to be deleted by specifying the full path (Example Storage/File.txt).

If the delete operation is successful, the function returns **TRUE**; on failure it returns **FALSE**.

Function parameters:

Name (STRING[32]) Filename to delete

La funzione ritorna:

(BOOL) **FALSE**: Execution error.
TRUE: Execution ok.

Error codes

If an error occurs the function returns **FALSE** and **SysGetLastError** can detect the error code.

- 9961100 Function used in task fast or slow.
- 9961150 Error in the file name declaration.
- 9961160 Directory not accessible by "Admin" user.
- 9961200 Error deleting the file.
- 9961990 Not yet Implemented in the simulator.

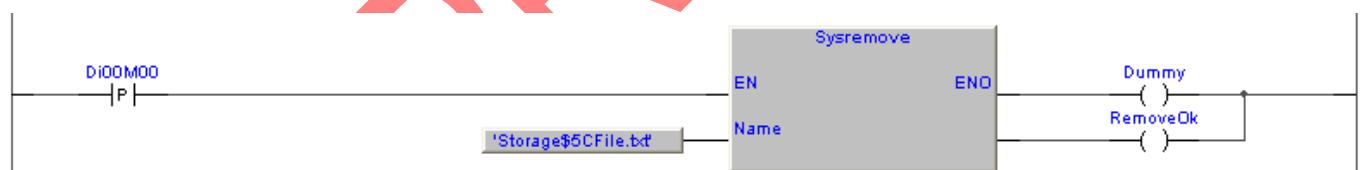
Examples

On rising edge of the **DI00M00** digital input, the **File.txt** file placed in the **Storage** directory, is deleted.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	RemoveOk	BOOL	Auto	No	FALSE	..	File remove Ok
2	Dummy	BOOL	Auto	No	FALSE	..	Dummy

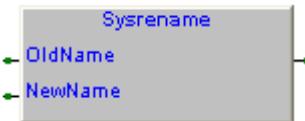
LD example



7.8.2 Sysrename, file rename

Type	Library
Function	XTarget_07_0

This function change the name of a file. In **OldName** the name of the file to be renamed must be defined by specifying the full path (Example Storage/OldFile.txt). In **NewName** the new name of the file must be defined by specifying the full path (Example Storage/NewFile.txt).



If the rename operation is successful, the function returns **TRUE**; on failure it returns **FALSE**.

Function parameters:

OldName (STRING[32]) Name of file to rename.

NewName (STRING[32]) New name to set to file.

Return value:

(BOOL) **FALSE**: Execution error.
TRUE: Execution ok.

Error codes

If an error occurs the function returns **FALSE** and **SysGetLastError** can detect the error code.

- 9960100 Function used in task fast or slow.
- 9960150 Error in the file **OldName** declaration.
- 9960160 **OldName** directory not accessible by "Admin" user.
- 9960170 Error in the file **NewName** declaration.
- 9960180 **NewName** directory not accessible by "Admin" user.
- 9960200 Error renaming the file.
- 9962990 Not yet Implemented in the simulator.

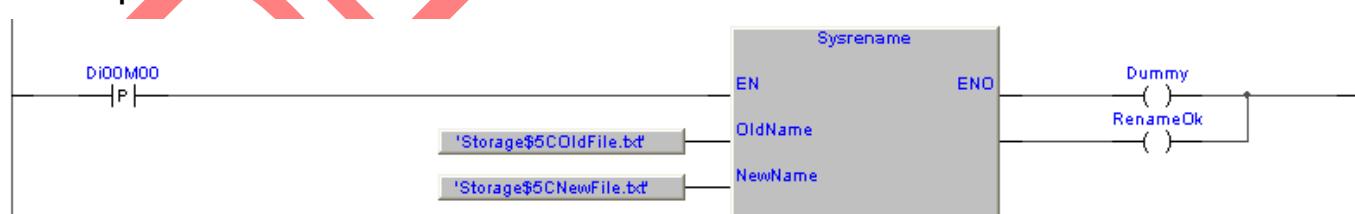
Examples

On rising edge of the **Di00M00** digital input, the **OldFile.txt** file placed in the **Storage** directory is renamed. New filename will be **NewFile.txt**.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Dummy	BOOL	Auto	No	FALSE	..	Dummy
2	RenameOk	BOOL	Auto	No	FALSE	..	File rename Ok

LD example



7.8.3 Sysfilelength, file length

Type	Library
Function	XTarget_07_0

This function returns the length in bytes of a file. **Name** defines the name of the file that you want to know the length specifying the full path (Example Storage/File.txt).

If the file is not present, the function returns -1.



Function parameters:

Name (STRING[32]) Name of the file that you want to know the length.

Return value:

(DINT) File lenght (Bytes). **EOF** if file not found.

Error codes

If an error occurs the function returns **EOF** and [SysGetLastError](#) can detect the error code.

9959100 Function used in task fast or slow.

9959990 Not yet Implemented in the simulator.

Examples

On rising edge of the **Di00M00** digital input, the lenght of **File.txt** file placed in the **Storage** directory is returned.

Definizione variabili

	Name	Type	Address	Array	Init value	Attribute	Description
1	FileSize	DINT	Auto	No	0	..	File size

LD example



7.8.4 Sysfseek, file seek

Type	Library
Function	XTarget_07_0

This feature allows you to change the position indicator of the stream connected to the **File** stream previously opened by the **Sysfopen** function.

Offset specifies the number of bytes from the origin, where should you put the position indicator.
Origin specifies the origin from which to move the position indicator.

The function returns the current value of the position indicator. In the case of positioning error, the position indicator remains unchanged and the function returns **EOF**.



Function parameters:

File (FILEP) Stream returned by **Sysfopen** function.

Offset (DINT) Number of bytes from the origin, where should you put the position indicator.

Origin (INT) Specifies the origin from which to move the position indicator, [FSeek origin definition](#).

Return value:

(INT) Actual indicator position. **EOF** is error.

Error codes

If an error occurs the function returns **EOF** and **SysGetLastError** can detect the error code.

9958100 Function used in task fast or slow.

9958990 Not yet Implemented in the simulator.

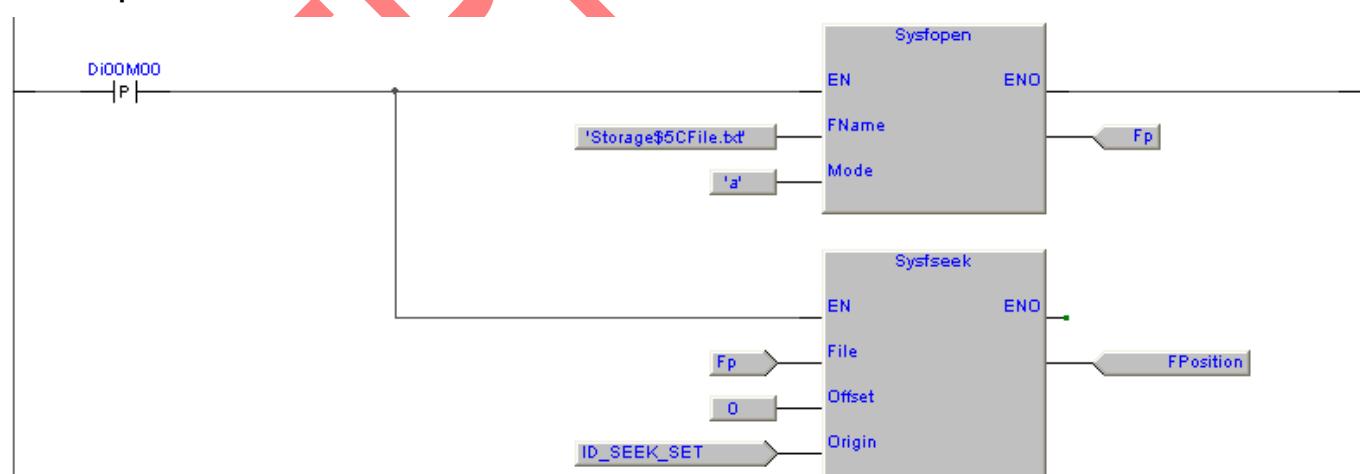
Examples

On rising edge of the **DI00M00** digital input, the position indicator will be placed at start of file **File.txt** placed in the **Storage** directory.

Defining variables

	Name	Type	Address	Array	InitValue	filelength	Attribute	Description
1	Fp	FILEP	Auto	No	0	File pointer
2	FPosition	DINT	Auto	No	0	File position

LD example

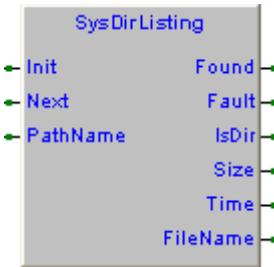


7.8.5 SysDirListing, directory listing

Type	Library
FB	XTarget_10_0

This function executes the directory listing of the directory defined in **PathName**. By setting the **Init** initializes the list of files and the first file found is returned. Ad every **Next** command a new file is find in the selected directory.

If a file is found the **Found** output is activated for a loop and its name is returned in **FileName**. Once all the files in the directory are listed, on a **Next** command is no longer activated the **Found** output and **FileName** is blanked. The output **IsDir** is set if the listed file is a subdirectory.



Init (BOOL)	By activating it's initialized the index of the files in the specified directory.
Next (BOOL)	By activating it's returned the name of the file pointed to by the index in the specified directory. The index is increased by focusing the next file.
PathName (STING[32])	Directory to be listed. A filter can be set (I.e 'Storage*.log').
Found (BOOL)	It is activated for a loop if on Init or Next command was found a new file to be listed.
Fault (BOOL)	It is activated for a loop on execution error.
IsDir (BOOL)	It is activated for a loop if the returned file is a subdirectory.
Size (UDINT)	File size (Bytes).
Time (UDINT)	Last file modification date, Epoch time (UTC).
FileName (STING[16])	File name with the extension.

Error codes

If an error occurs the **Fault** output is activated and [SysGetLastError](#) can detect the error code.

- 9952050 Function block allocation error.
- 9952060 Relocatable memory space full. You can not run the FB.
- 9952070 Function block version error.
- 9952100 FB used in task fast or slow
- 9952200 Directory listing execution error.
- 9952990 Not yet Implemented in the simulator.

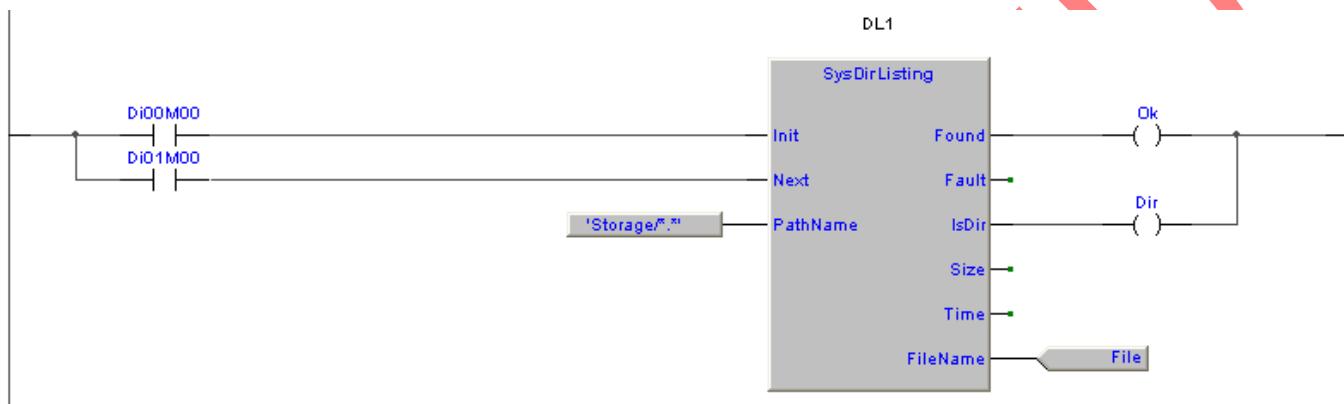
Examples

By activating the digital input ***Di00M00*** the files pointer in the ***Storage*** directory is initialized. At each activation of the digital input ***Di01M00*** in ***File*** is returned a new file name in the directory. If a new file is returned, ***Ok*** is activated for a loop, if it's a subdirectory will ***Dir*** is activated.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Dir	BOOL	Auto	No	FALSE	..	Is a directory
2	Ok	BOOL	Auto	No	FALSE	..	File found on Next command
3	File	STRING	Auto	[16]		..	File name
4	DL1	SysDirListing	Auto	No	0	..	FB directory listing

LD example



DEPRECATION

7.9 Functions and FBs for serial port management

The SlimLine systems have a number of serial ports related to the version of the product. The serial ports are identified with **COMx**, a type name where x stay for the number of serial port.

For the correspondence between the port number and the physical connector associated with it, see the hardware manual of the product.

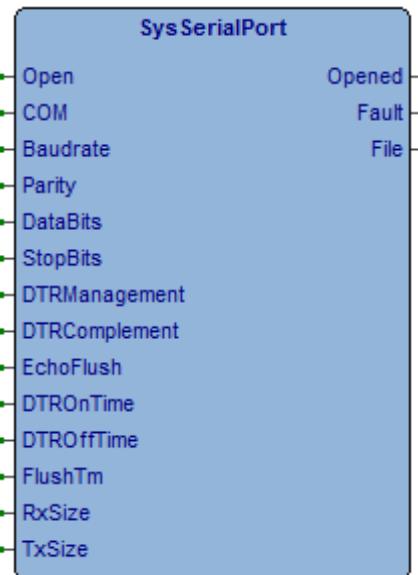
DEPRECATED

7.9.1 SysSerialPort, manage serial port

Type	Library
FB	XTarget_12_0

Questo blocco funzione gestisce la comunicazione su porta seriale. Occorre definire in COM la porta da gestire fornendo tutti i parametri di configurazione. Attivando il comando **Open** la porta viene aperta se non ci sono problemi viene attivato **Opened** e sull'uscita **File** viene ritornato lo stream da utilizzarsi per lo scambio dati sulla porta.

Se ci sono errori nei parametri o con la porta definita viene generato **Fault**.



Open (BOOL)	Comando apertura porta seriale.
COM (STRING[12])	Flusso dati stream ritornato dalla funzione Sysfopen .
Baudrate (UDINT)	Valore di baud rate porta seriale (da 300 a 115200 baud)
Parity (STRING[1])	Tipo di parità, valori possibili "E" pari, "O" dispari, "N" nessuna.
DataBits (USINT)	Numero di bit frame dato, valori possibili 7, 8.
StopBits (USINT)	Numero di bit di stop, valori possibili 1, 2.
DTRManagement (USINT)	Modo di gestione del segnale DTR sulla porta seriale, vedi definizione .
DTRComplement (BOOL)	FALSE : DTR normale, TRUE : DTR complementato.
EchoFlush (BOOL)	FALSE : I dati trasmessi sono ritornati in ricezione. TRUE : I dati trasmessi sono ignorati.
DTROnTime (UINT)	Tempo di attesa dopo attivazione segnale DTR prima di trasmissione caratteri (mS).
DTROffTime (UINT)	Tempo di attesa dopo trasmissione ultimo dato prima e disattivazione segnale DTR (mS).
FlushTm (UINT)	Tempo di flush dati, se non sono caricati dati sullo stream dopo il tempo definito i dati presenti vengono automaticamente inviati (mS) (Impostare 0).
RxSize (UINT)	Dimensione buffer ricezione dati (Impostare 0).
TxSize (UINT)	Dimensione buffer trasmissione dati (Impostare 0).
Opened (BOOL)	Attivo se porta seriale aperta.
Fault (BOOL)	Attivo se errore gestione
File (FILEP)	Stream di I/O. Viene valorizzato su apertura porta seriale.

Codici di errore

In caso di errore la funzione torna **FALSE** e con [**SysGetLastError**](#) è possibile rilevare il codice di errore.

- 9946050 Errore allocazione blocco funzione.
- 9946060 Terminato spazio memoria rilocabile, non è possibile eseguire l'FB.
- 9946070 Errore versione blocco funzione.
- 9946100 Dimensione buffers o flush time errati.
- 9946110 Errore impostazione porta seriale.
- 9946120 Errore baudrate.
- 9946121 Errore parity.
- 9946122 Errore data bits.
- 9946123 Errore stop bits
- 9946124 Errore DTR
- 9946990 Non implementata nel simulatore.

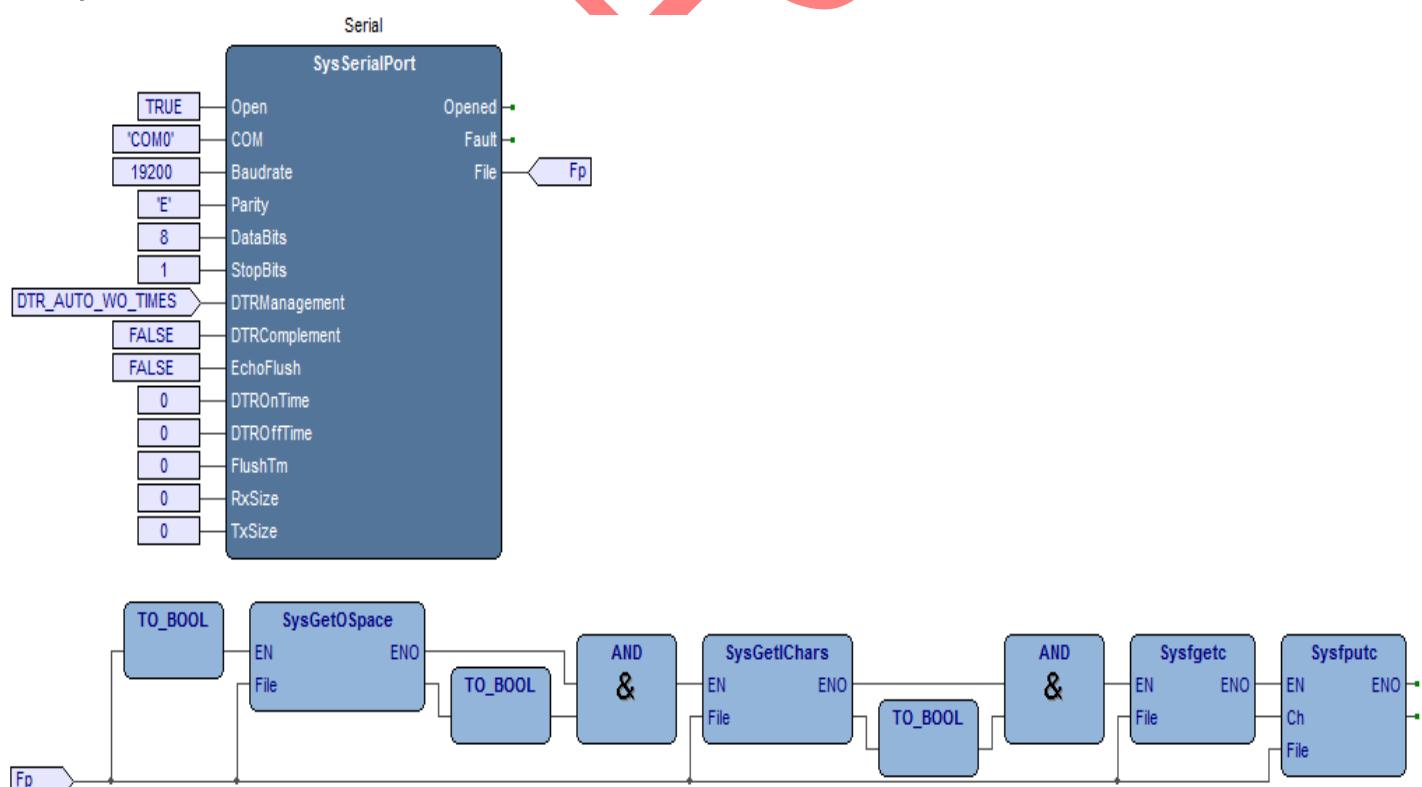
Esempi

Nell'esempio è attivato un server in ascolto sulla porta seriale. Connnettendosi in seriale alla COM0 inviando un carattere se ne riceve l'echo.

Definizione variabili

	Name	Type	Address	Array	Init value	Attribute	Description
1	Fp	FILEP	Auto	No	File pointer array
2	Serial	SysSerialPort	Auto	No	

Esempio FBD (PTP119B000, FBD_SerialEcho)



7.9.2 SysGetSerialMode, get serial mode

Type	Library
Function	XTarget_07_0

This function returns the communication mode of the serial port connected to the **File** parameter previously opened by the [Sysfopen](#) function.

In the **Mode** parameter must define the address of the [**SYSSERIALMODE**](#) data structure where will be transferred the actual serial port settings. The function returns FALSE on error.



Function parameter:

Mode (@SYSSERIALMODE) Address of [**SYSSERIALMODE**](#) structure where to copy the serial port settings.

File (FILEP) Stream returned by [Sysfopen](#) function.

Return value:

(BOOL) **FALSE**: Execution error. **TRUE**: Execution ok.

Error codes

If an error occurs the function returns **FALSE** and [SysGetLastError](#) can detect the error code.

9995010 **File** value not defined.

9995020 **SYSSERIALMODE** structure address not correct. Verify **Mode**.

9995100 ÷ 1 Execution error.

9995990 Not yet Implemented in the simulator.

Examples

On rising edge of the **Di00M00** digital input, the settings of serial port **COM0** are saved in the **Sm** variable and the **Do00M00** logic output is activated.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Fp	FILEP	Auto	No	0	..	Port COM0 file pointer
2	Sm	SYSSERIALMODE	Auto	No	0	..	Serial mode data struct
3	Pulse	BOOL	Auto	No	FALSE	..	Pulse flag

ST example (PTP116A300, ST_SysGetSerialMode)

```

(* Here the COM0 port is opened in read/write. *)

IF (Fp = NULL) THEN
  Fp:=Sysfopen('COM0', 'rw'); (* Port COM0 file pointer *)
END_IF;

(* Check if the COM0 port is open. *)

IF (Fp <> NULL) THEN
  (* Check if input is activated. *)

  IF (Di00M00 <> Pulse) THEN
    Pulse:=Di00M00; (* Pulse flag *)

    (* On input raising edge the serial mode is read. *)

    IF (Di00M00) THEN
      Do00M00:=SysGetSerialMode(ADR(Sm), Fp);
    END_IF;
  END_IF;
END_IF;
  
```

By putting the Sm variable (of type **SYSSERIALMODE**) in the watch, we can see the values of all its members as shown in the figure. In this case it appears the default configuration **115200,e,8,1**.

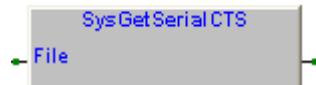
Symbol	Value	Type
- Sm	-	SYSSERIALMODE
Baudrate	115200	UDINT
Parity	'E'	STRING
DataBits	8	USINT
StopBits	1	USINT
DTRManagement	2	USINT
DTRComplement	FALSE	BOOL
EchoFlush	FALSE	BOOL
DTROnTime	0	UINT
DTROffTime	0	UINT

DEPRECATELY

7.9.3 SysGetSerialCTS, get serial CTS signal status

Type	Library
Function	XTarget_07_0

This function returns the status of the **CTS** signal of the serial port specified by **File** previously opened by the **Sysfopen** function.



Function parameters:

File (FILEP) Serial port stream returned by **Sysfopen** function.

Return value:

(BOOL) **FALSE**: CTS deactive. **TRUE**: CTS active.

Error codes

If an error occurs the function returns **FALSE** and **SysGetLastError** can detect the error code.

9993010 **File** value not defined.

9993990 Not yet Implemented in the simulator.

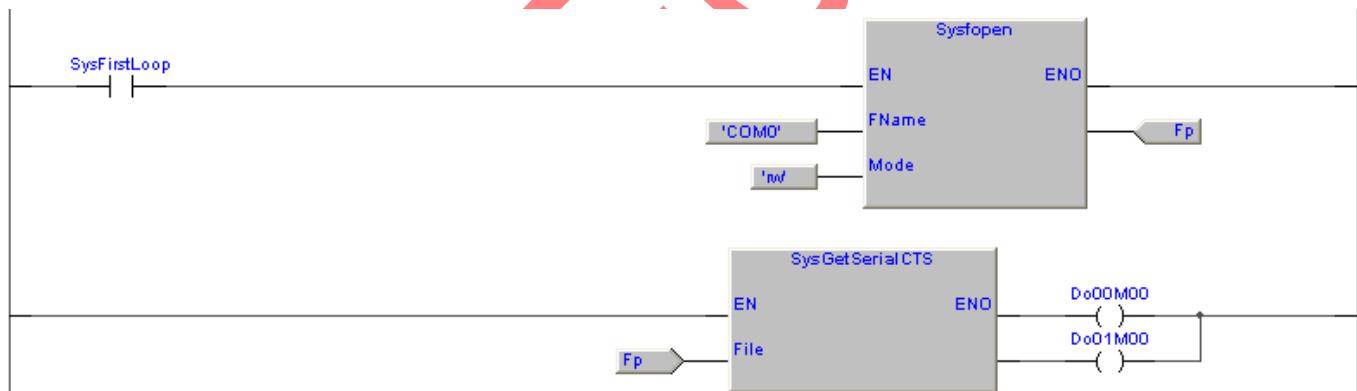
Examples

The **CTS** status of **COM0** serial port is copied to **Do01M00** output.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Fp	FILEP	Auto	No	0	..	File pointer

LD example



7.9.4 SysSetSerialDTR, set DTR signal status

Type	Library
Function	XTarget_07_0

This function sets the state of the **DTR** signal of the serial port specified by **File** previously opened by the **Sysfopen** function.

In order to handle the DTR signal, the serial port must be set with the **DTRManagement** of **SYS SERIAL MODE** structure to **DTR_OFF**.



Function parameters:

Status (BOOL) DTR status to set on serial port.

File (FILEP) Serial port stream returned by **Sysfopen** function.

Return value:

(BOOL) **FALSE**: Execution error. **TRUE**: Execution ok.

Error codes

If an error occurs the function returns **FALSE** and **SysGetLastError** can detect the error code.

9992010 **File** value not defined.

9992990 Not yet Implemented in the simulator.

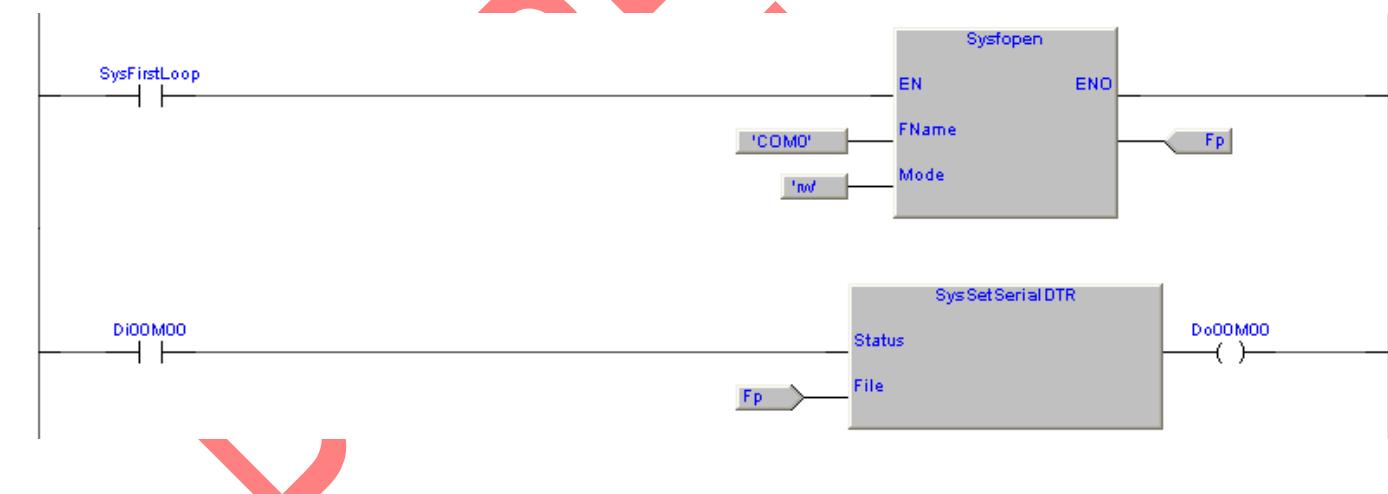
Examples

The status of **Di00M00** input is transferred to the DTR of serial port **COM0**.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Fp	FILEP	Auto	No	0	..	File pointer

LD example



7.10 Functions and FBs for CAN bus management

The Controller Area Network, also known as CAN-bus is a standard serial multicast fieldbus, to connect various electronic control units (ECUs). The CAN was specifically designed to work without any problems even in very noisy environments by the presence of electromagnetic waves and can be used as a means of transmission line to the potential difference as the RS-485 balanced. The immunity to electromagnetic interference can be further increased by using twisted pair cables.

Although initially applied in the automotive sector as bus line for cars, it's currently used in many industrial embedded applications, which requires a high level of noise immunity. The bit rate can reach 1 Mbit/s for long networks less than 40 m. Slower speeds allow you to reach greater distances (eg. 125 kbit/s to 500 m). The CAN communication protocol is standardized in ISO 11898-1 (2003).

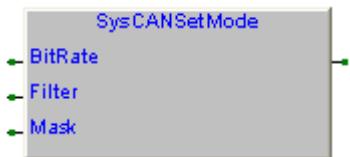
DEPRECATED

7.10.1 SysCANSetMode, set the CAN controller mode

Type	Library
Function	XTarget_07_0

This function sets the CAN controller mode. It's possible to define the bit rate, the packet filter and mask acceptance.

The function returns **TRUE** if successful, **FALSE** on error.



Function parameters:

BitRate (USINT) CAN bus bit rate definition, [CAN bit rate definition](#).

Filter (UDINT) CAN packet acceptance filter.

Mask (UDINT) CAN packet acceptance mask.

Return value:

(BOOL) **TRUE**: Successfully executed.
FALSE: If an error occurs, such as incorrect parameters.

Error codes

If an error occurs the function returns **FALSE**, using the [SysGetLastError](#) the error code can be detected.

9957005 Function not supported.

9957990 Not yet Implemented in the simulator.

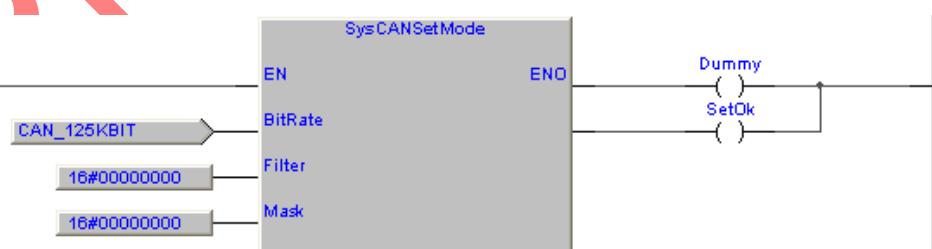
Examples

It's shown a simple program that set the CAN controllers with bit rates to 125 kbps. All incoming packets are received by the controller.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Dummy	BOOL	Auto	No	FALSE	..	Dummy variable
2	SetOk	BOOL	Auto	No	FALSE	..	CAN set mode Ok

LD example



7.10.2 SysIsCANRxTxAv, checks if CAN Rx or Tx is available

Type	Library
Function	XTarget_07_0

This function controls:

Select: FALSE: If at least one CAN message is present in the receive buffer.

Select: TRUE: If there is room to transmit a CAN message.



Function returns TRUE if the selected condition is true.

Function parameters:

Select (BOOL) **FALSE:** If at least one CAN message is present in the receive buffer.
TRUE: If there is room to transmit a CAN message.

Return value:

(BOOL) **TRUE:** Selected condition is true.

Error codes

If an error occurs the function returns **FALSE**, using the [SysGetLastError](#) the error code can be detected.

9956005 Function not supported.

9956990 Not yet Implemented in the simulator.

Examples

It is showed a simple program that makes the control if a CAN message is received, read it and send the structure of the received message to the serial port **COM0**.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Fp	FILEP	Auto	No	0	..	Port COM0 file pointer
2	CANMsg	SYSCANMESSAGE	Auto	No	0	..	CAN message
3	NrOfChars	INT	Auto	No	0	..	Number of printed chars

ST example

```
(* Here the COM0 port is opened in read/write. *)
IF (Fp = NULL) THEN
  Fp:=Sysfopen('COM0', 'rw'); (* Port COM0 file pointer *)
END_IF;

(* Here check if a CAN message is available and receive it. *)
IF (SysIsCANRxTxAv(FALSE)) THEN
  IF (SysCANRxMsg(16#00000000, 16#00000000, ADR(CANMsg))) THEN
    NrOfChars:=SysVarfprintf(Fp, 'Length:%04d$r$n', USINT_TYPE, ADR(CANMsg.Length));
    NrOfChars:=SysVarfprintf(Fp, 'MsgID:%04d$r$n', UDINT_TYPE, ADR(CANMsg.MsgID));
    NrOfChars:=SysVarfprintf(Fp, 'Data[0]:%02X$r$n', UDINT_TYPE, ADR(CANMsg.Data[0]));
  END_IF;
END_IF;
```

7.10.3 SysCANRxMsg, receives a CAN message

Type	Library
Function	XTarget_07_0

This function receives a CAN message and transfers it to the variable whose address is defined in the **Msg**. It is possible to define a **Mask** and an **ID** to receive only CAN messages you want.

The function search on the stack of messages a message whose **ID** in logic AND with **Mask**, match the **ID** in logic AND with **Mask**. The function returns TRUE if the message was received.



Function parameters:

- Mask** (UDINT) Mask code.
ID (UDINT) Message ID.
Msg (@SYSCANMESSAGE) Received message buffer address.

Return value:

- (BOOL) **TRUE**: Message received.

Error codes

If an error occurs the function returns **FALSE**, using the [SysGetLastError](#) the error code can be detected.

9955005 Function not supported.

9955990 Not yet Implemented in the simulator.

Examples

It is showed a simple program that performs the reception of any CAN message and performs sending out to the serial port **COM0** of the structure of the received message.

Defining variables

	Name	Type	Address	Array	Initvalue	Attribute	Description
1	Fp	FILEP	Auto	No	0	..	Port COM0 file pointer
2	CANMsg	SYSCANMESSAGE	Auto	No	0	..	CAN message
3	NrOfChars	INT	Auto	No	0	..	Number of printed chars

ST example

```

(* Here the COM0 port is opened in read/write. *)

IF (Fp = NULL) THEN
  Fp:=Sysopen('COM0', 'rw'); (* Port COM0 file pointer *)
END_IF;

(* Here receive a CAN message. *) */

IF (SysCANRxMsg(16#3FFFFFFF, 16#00000000, ADR(CANMsg))) THEN
  NrOfChars:=SysVarfprintf(Fp, 'Length:%04d$r$n', USINT_TYPE, ADR(CANMsg.Length));
  NrOfChars:=SysVarfprintf(Fp, 'MsgID:%04d$r$n', UDINT_TYPE, ADR(CANMsg.MsgID));
  NrOfChars:=SysVarfprintf(Fp, 'Data[0]:%02X$r$n', UDINT_TYPE, ADR(CANMsg.Data[0]));

END_IF;
  
```

7.10.4 SysCANTxMsg, transmit a CAN message

Type	Library
Function	XTarget_07_0

This function sends a CAN message. We need to create the message and to pass its address in **Msg**.

Function returns TRUE if the message was sent.

Function parameters:

Msg (@SYSCANMESSAGE) Buffer address of message to send.

Return value:

(BOOL) **TRUE**: Message sent.

Error codes

If an error occurs the function returns **FALSE**, using the [SysGetLastError](#) the error code can be detected.

9954005 Function not supported.

9954990 Not yet Implemented in the simulator.

Examples

It is showed a simple program that performs the transmission of a CAN message.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	CANMsg	SYSCANMESSAGE	Auto	No	0	..	CAN message
2	TxOk	BOOL	Auto	No	FALSE	..	Transmission Ok

ST example

(* Here check if there is a space in Tx buffer and send a CAN message. *)

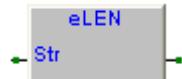
```
IF (SysIsCANRxTxAv(TRUE)) THEN
  CANMsg.RmReq:=FALSE; (* eFALSE:Data frame, eTRUE:Remote request *)
  CANMsg.Length:=2; (* Data length *)
  CANMsg.MsgID:=16#00000000; (* Message ID (FF:Bit 31) (11 or 29 Bit) *)
  CANMsg.Data[0]:=16#01; (* Message data *)
  CANMsg.Data[1]:=16#00; (* Message data *)
  TxOk:=SysCANTxMsg(ADR(CANMsg)); (* Transmission Ok *)
END_IF;
```

7.11 Functions and FBs for string management

7.11.1 eLEN, string length

Type	Library
Function	ePLCStdLib_B000

This function returns the length (expressed in number of characters) of the string defined in **Str**.



Function parameters:

Str (@USINT) Pointer to string to return lenght.

Return value:

(INT) Number of characters present in the string.

Examples

It is calculated the length of the string '**Hello!**' and the number of characters in the string is passed in the **Length** variable. The calculation result is **6**.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Length	INT	Auto	No	0	..	String length

LD example



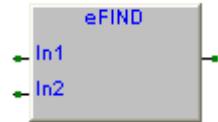
7.11.2 eFIND, string find

Type	Library
Function	ePLCStdLib_B000

This function searches the position of the first occurrence of the string **In2** in **In1**. If no occurrence is found, the function returns **0**.

If the string **In2** is found in the string **In1**, then its position is returned.

Example: **eFIND(In1:='abcd', In2:='bc')**. The result is **2**.



Function parameters:

In1 (@USINT) Pointer to string where to make search.

In2 (@USINT) Pointer to string to search.

Return value:

(INT) Position of the first occurrence of **In2** in **In1**. **0** if string not found.

Examples

It is searched the string '**lo**' in string '**Hello world!**'. The location found is **4** and is transferred to the variable **Position**.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Position	INT	Auto	No	0	..	StrToFind position
2	StrSource	STRING	Auto	[32]		..	String were looking
3	StrToFind	STRING	Auto	[32]		..	String to find

ST example

```
(* Find the position where is StrToFind in StrSource. *)
StrSource:='Hello world!';
StrToFind:='lo';

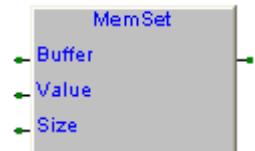
Position:=eFIND(ADR(StrSource), ADR(StrToFind));
```

7.11.3 MemSet, memory set

Type	Library
Function	eLLabUtyLib_C030

This function set **Size** bytes of **Buffer** with the value defined in **Value**.

The function returns the number defined in **Size**.



Function parameters:

Buffer (@USINT) Pointer to memory buffer where to set **Value**.

Value (USINT) Value to set to memory buffer.

Size (UDINT) Number of bytes to set with **Value** starting from **Buffer**.

Return value:

(UDINT) **Size** Value.

Examples

Set to zero all the bytes of the string **StrBuffer** writing the value **0** in its entire length.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	StrBuffer	STRING	Auto	[32]	String buffer
2	RetVal	UDINT	Auto	No	0	..	Return value

ST example

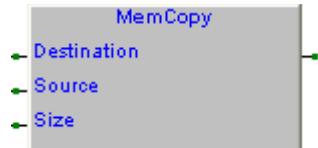
```
(* The 'StrBuffer' variable is set to '0'. *)
RetVal:=MemSet(ADR(StrBuffer), 0, 32); (* Return value *)
```

7.11.4 MemCopy, memory copy

Type	Library
Function	eLLabUtyLib_C030

This function copies the number of bytes defined in **Size** starting from the memory area pointed to by **Source**, to the area that starts at **Destination**.

The function returns the number defined in **Size**.



Function parameters:

Destination (@USINT) Pointer to destination buffer.
Source (@USINT) Pointer to source buffer.
Size (UDINT) Number of byte to be transferred

Return value:

(UDINT) **Size** Value.

Examples

It's executed the copy of the **SString** in the **DString**. At the end of the resulting string will be "HelloHello.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	RetVal	UDINT	Auto	No	0	..	Return value
2	DString	STRING	Auto	[32]		..	Destination string
3	SString	STRING	Auto	[8]	Hello	..	Source string

ST example

(* The 'SString' is copied twice into 'DString'. *)

```

RetVal:=Memcpy(ADR(DString), ADR(SString), SIZEOF(SString)); (* Return value *)
RetVal:=Memcpy(ADR(DString)+LEN(DString), ADR(SString), SIZEOF(SString)); (* Return value *)
  
```

7.11.5 SysVarsnprintf, variable print to string

Type	Library
Function	XTarget_08_0

This function transfers in **String** the formatted variable. The formatted value returned in the String variable, can not exceed the length defined in **Size**.

The **Format** string specifies the format in which to print the variable. **VarType** is the variable type and **VarAdd** is its address.

The function returns the number of characters transferred to the **String** variable. **EOF** on error.

Function parameters:

String (@USINT)	Pointer to the string which must contain the printed results.
Size (UINT)	Number of characters to be transferred into the String variable. The definite number is inclusive of the code at the end of string '\0'. If the length of output string exceeds the Size bytes, it is truncated to the number of bytes indicated.
Format (STRING[80])	It has two types of subjects: ordinary characters that are copied into the String variable output, and conversion specifications, denoted by the symbol percent (%) and a character that specifies the format in which to print the defined variable.
VarType (USINT)	Variable type as indicated in the table variable types definition .
VarAdd (UDINT)	Address of variable.

Return value:

(INT)	Number of code characters including the end of string '\0' transferred to the String variable. EOF : Error executing.
-------	--

Error codes

If an error occurs the function returns **EOF** and [SysGetLastError](#) can detect the error code.

9997100 Variable type not managed. Check **VarType**.

9997200 The **Size** value, limits the format of the out string.

Examples

On rising edge of **Di00M00** input, the **Counter** variable is increased and its value is formatted in **StringOut**. The value in **StringOut** is then sent to the serial port **COM0**. **NrOfChars** is loaded with the number of characters formatted in **StringOut** and sent out to the serial port.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Fp	FILEP	Auto	No	0	..	Port COM0 file pointer
2	Pulse	BOOL	Auto	No	FALSE	..	Pulse flag
3	Counter	UDINT	Auto	No	0	..	Counter
4	NrOfChars	INT	Auto	No	0	..	Number of printed chars
5	StringOut	USINT	Auto	[0..31]	32(0)	..	String output
6	i	INT	Auto	No	0	..	Auxiliary counter
7	Ch	INT	Auto	No	0	..	Character written

ST example (PTP116A100, ST_SysVarsnprintf)

(* Here the COM0 port is opened in read/write. *)

```
IF (Fp = NULL) THEN
    Fp:=Sysfopen('COM0', 'rw'); (* Port COM0 file pointer *)
END_IF;
```

(* Check if the COM0 port is open. *)

```
IF (Fp <> NULL) THEN
```

```

(* Check if input is activated. *)

IF (Di00M00 <> Pulse) THEN
    Pulse:=Di00M00; (* Pulse flag *)

(* On input raising edge the counter value is printed. *)

IF (Di00M00) THEN
    Counter:=Counter+1; (* Counter *)
    NrOfChars:=SysVarnprintf(ADR(StringOut), 32, 'Counter:%04d$r$', UDINT_TYPE, ADR(Counter));

    (* Copy the printed result to serial port. *)

    FOR i:=0 TO NrOfChars DO Ch:=Sysfputc(TO_INT(StringOut[i]), Fp); END_FOR;
END_IF;
END_IF;
END_IF;

```

In this example, the merge is performed between prints value of two variables. Execute merge can be very useful to have a single string containing printed value of several variables.

By putting the **Result** variable in debug, we will see the string **Var[0]:12 Var[1]:34**. Having blocked the printing to 12 characters the value of **Var[0]** will be printed correctly to a maximum of 4 digits (7 string characters, 4 string terminator character '\0'). For values of **Var[0]** greater than 9999 will no longer print the least significant digits.

The value of **Var[1]** will be printed immediately after the value of **Var[0]**. Note the decreased offset of 1 to overwrite the string terminator '\0'. To guarantee the printing of up to 4 for **Var[1]**, has been used a **Size** of 13. The string of text starts with a space character to separate it from the print out of the previous variable.

Defining variables

	Name	Type	Address	Array	Initvalue	Attribute	Description
1	NrOfChars	INT	Auto	No	0	..	Number of printed chars
2	Var	UDINT	Auto	[0..1]	12,34	..	Variables
3	Result	STRING	Auto	[32]		..	

ST example

```

(* ----- *)
(* EXECUTE A VARIABLES PRINT MERGE *) 
(* ----- *)
(* Print the variable values, merging them into a single string. *)

NrOfChars:=SysVarnprintf(ADR(Result), 12, 'Var[0]:%d', UDINT_TYPE, ADR(Var[0]));
NrOfChars:=SysVarnprintf(ADR(Result[LEN(Result)]), 13, ' Var[1]:%d', UDINT_TYPE, ADR(Var[1]));

(* [End of file] *)

```

7.11.6 SysLWVarsnprintf, variable print to string with append

Type	Library
Function	XTarget_12_0

Questa funzione trasferisce in **String** la stampa formattata di una variabile. Il valore stampato ritornato nella variabile stringa non può superare la lunghezza definita in **Size**.

La stringa **Format** specifica il formato con il quale stampare la variabile. Mentre in **VarType** è indicato il tipo di variabile ed in **VarAdd** il suo indirizzo.

A differenza dalla funzione [SysVarsnprintf](#) la stampa formattata viene aggiunta al termine del contenuto di **String**. Questo permette di concatenare più stampe in una unica stringa. La stampa viene troncata nel caso in cui la lunghezza totale della stringa superi **Size**.

La funzione ritorna il numero di caratteri totali presenti nella variabile **String**. EOF in caso di errore.

Parametri funzione:

String (@USINT)	Pointer all'array dove deve essere trasferito il risultato della stampa.
Size (UINT)	Numero di caratteri da trasferire nella variabile String . Il numero definito è comprensivo del codice di fine stringa '\0'. Se la lunghezza della stringa di output supera il limite di Size byte, viene troncata al numero di byte indicato.
Format (STRING[80])	Ha due tipi di argomenti, i caratteri ordinari che vengono copiati nella variabile String di uscita, e le specifiche di conversione, contraddistinte dal simbolo percentuale (%) e da un carattere che specifica il formato con il quale stampare la variabile definita.
VarType (USINT)	Tipo variabile, come indicato nella tabella Variable types definition .
VarAdd (UDINT)	Indirizzo variabile.

La funzione ritorna:

(INT)	Numero di caratteri totali comprensivo codice fine stringa '\0' presenti nella variabile String . EOF : Errore esecuzione.
-------	---

Codici di errore

In caso di errore la funzione torna con **EOF** e con [SysGetLastError](#) è possibile rilevare il codice di errore.

9993100 Tipo variabile non gestito, controllare **VarType**.

9932110 Lunghezza stringa è maggiore di **Size** (Impossibile eseguire la funzione).

9993200 Il valore di **Size** limita la formattazione della stringa in uscita.

Esempi

Vedere esempi della funzione [SysVarsnprintf](#).

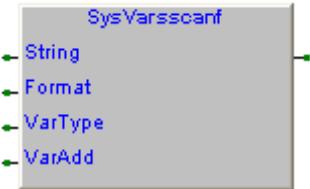
7.11.7 SysVarsscanf, extracts values from string

Type	Library
Function	XTarget_08_0

This function reads the **String**, and interprets it according to the specified **Format**.

The **Format** string specifies the format in which to interpret the variable. **VarType** indicates the variable type and **VarAdd** its address.

The function returns TRUE if the variable was found, in other case FALSE.



Function parameters:

String (@USINT) Pointer to string to read.

Format (STRING[80]) It has two types of subjects: ordinary characters that are checked into the **String** input variable, and conversion specifications, denoted by the symbol percent (%) and a character that specifies the format in which to print the defined variable.

VarType (USINT) Variable type as indicated in the table [variable types definition](#).

VarAdd (UDINT) Variable address.

Return value:

(BOOL) **TRUE**: Variable value acquired.

Error codes

If an error occurs the function returns **FALSE** and [SysGetLastError](#) can detect the error code.

9999100 Variable type not managed. Check **VarType**.

Examples

It is showed a program that reads a **InputString** string init with text Value:123. On **Di00M00** input rising edge, three different functions are exected, all on **InputString** string but with different definitions for **Format**. The first two are successful and variables **Variable[0]** and **Variable[1]** will be set with the value **123**. The third will fail and the variable **Variable[2]** will be reset.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Pulse	BOOL	Auto	No	FALSE	..	Pulse flag
2	Variable	UDINT	Auto	[0..2]	3(0)	..	Variable
3	InputString	STRING	Auto	[32]	Value:123	..	Input string
4	Result	BOOL	Auto	[0..2]	3(0)	..	Function result

ST example (PTP116A200, ST_SysVarsscanf)

```

(* Check if input is activated. *)

IF (Di00M00 <> Pulse) THEN
    Pulse:=Di00M00; (* Pulse flag *)

    IF (Di00M00) THEN
        Result[0]:=SysVarsscanf(ADR(InputString), 'Value:%d', UDINT_TYPE, ADR(Variable[0]));
        Result[1]:=SysVarsscanf(ADR(InputString)+6, '%d', UDINT_TYPE, ADR(Variable[1]));
        Result[2]:=SysVarsscanf(ADR(InputString), '%d', UDINT_TYPE, ADR(Variable[2]));
    END_IF;
END_IF;
  
```

7.11.8 SysFWVarsscanf, extracts values from string with find

Type	Library
Function	XTarget_12_0

Questa funzione legge la stringa **String** e ne interpreta il contenuto basandosi sul parametro **Format**.

La stringa **Format** specifica il formato con il quale interpretare la variabile, in **VarType** è indicato il tipo di variabile ed in **VarAdd** il suo indirizzo.

A differenza dalla funzione [SysVarVarsscanf](#) l'interpretazione del contenuto di **String** avviene dopo avere cercato eventuali riferimenti definiti in **Format**.

La funzione ritorna TRUE se valore variabile trovato, in caso contrario FALSE.

Parametri funzione:

String (@USINT) Pointer alla stringa da leggere.

Format (STRING[80]) Ha due tipi di argomenti, i caratteri ordinari che vengono controllati nella variabile **String** da leggere e le specifiche di conversione, contraddistinte dal simbolo percentuale (%) e da un carattere che specifica il formato con il quale stampare la variabile definita.

VarType (USINT) Tipo variabile, come indicato nella tabella [Variable types definition](#).

VarAdd (UDINT) Indirizzo variabile.

La funzione ritorna:

(INT) Numero di conversioni effettuate

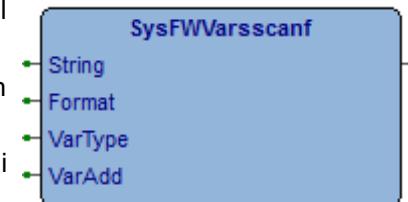
Codici di errore

In caso di errore la funzione torna con **FALSE** e con [SysGetLastError](#) è possibile rilevare il codice di errore.

9999100 Tipo variabile non gestito, controllare **VarType**.

Esempi

Fare riferimento agli esempi dalla funzione [SysVarVarsscanf](#).



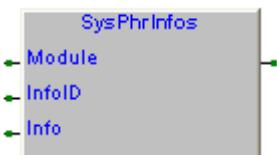
7.12 Functions and FBs for extension modules management

7.12.1 SysPhrInfos, get infos from peripheral modules

Type	Library	Version
Funzione	Embedded	XTarget 07.0

This function executes the acquisition of informations from the peripheral modules. The information indicated by **InfoID** is read from the **Module** extension module and is copied into the string variable whose address is passed by **Info**.

Function return **TRUE** if ok, **FALSE** in other cases.



Function parameters:

Module (USINT) You must specify the module address from which to get informations (range from 0 to 15). The value 0 indicates the first extension module, 1 for the second and so on.

InfoID (USINT) You must specify the ID of required information.

Value	Description
0	Return the product code
1	Return the software code

Info (STRING[10]) Address of variable where to copy the read information.

Return value:

(BOOL) **FALSE**: Execution error. **TRUE**: Execution ok.

Error codes

If an error occurs the function returns **FALSE** and [SysGetLastError](#) can detect the error code.

9990100 Addressed **Module** not present.

9990110 **InfoID** not correct.

9990200 The required information is not supported by module.

9990210 Error during request of information from module.

9990990 Not yet Implemented in the simulator.

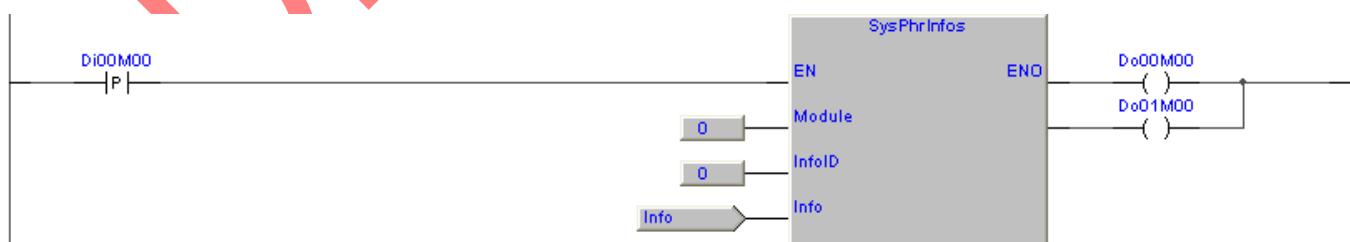
Examples

It is showed a program that reads the product code from extension module with address 0. The returned code is transferred into the **Info** variable. The return of the function is transferred to the **Do01M00** output.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Info	STRING	Auto	[10]	..		Info returned from module

LD example



7.12.2 SysGetPhrDI, get peripheral digital input

Type	Library	Version
FB	Embedded	XTarget 07.0

This function block performs the acquisition of digital inputs from the peripheral modules. The function block returns the status of digital inputs from the module specified by **Address** and using the **Mode** defined.

To acquire the digital inputs on the CPU module, you must define 255 as **Address** and DI_8_LL as **Mode**.



Address (USINT)	You must specify the module address from which to acquire the digital inputs (range 0 to 255). A value of 0 indicates the first extension module, 1 for the second and so on. The address 255 indicates the CPU module.
Mode (USINT)	You must specify the mode of acquisition of the digital inputs, refer to Digital input mode .
Done (BOOL)	Data acquired. Activated if the digital inputs acquisition, has terminated.
Fault (BOOL)	Acquisition error. It is activated when an error occurs in the sequence of acquisition.
Value (UDINT)	Returns the status of digital inputs acquired.

Error codes

If an error occurs, the **Fault** output is activated and [SysGetLastError](#) can detect the error code.

- 9985050 Function block allocation error.
- 9985070 Function block version error.
- 9985100 The module addressed with **Address** is not present.
- 9985110~24 The acquisition **Mode** is not correct.
- 9985200~13 Error during reading of module peripheral input.

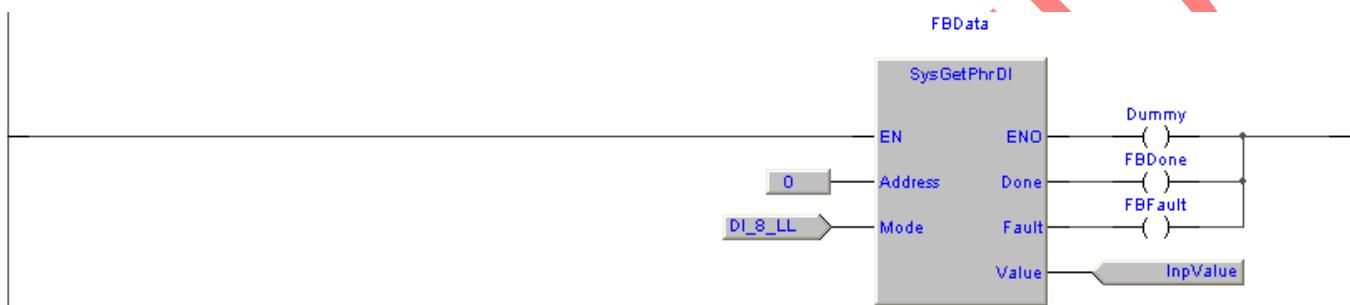
Examples

It acquired the status of the 8 inputs low (from inp 0 to Inp 7) of the module with address 0. If data valid, the **FBDone** variable is activated; if acquisition error, the **FBFault** variable is activated. The value obtained in the range 0x00 to 0xFF, is transferred into the **InpValue** variable.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	FBData	SysGetPhrDI	Auto	No	0	..	FB SysGetPhrDI data
2	InpValue	UDINT	Auto	No	0	..	Digital input value
3	Dummy	BOOL	Auto	No	FALSE	..	Dummy variable
4	FBDone	BOOL	Auto	No	FALSE	..	FB done
5	FBFault	BOOL	Auto	No	FALSE	..	FB fault

LD example (PTP116A100, LD_SysGetPhrDI)



IL example (PTP116A100, IL_SysGetPhrDI)

(* Read Inp 0 to 7 from module with address 0. *)

```

LD 0
ST FBData.Address

LD DI_8_LL
ST FBData.Mode

CAL FBData

LD FBData.Value
ST InpValue

```

ST example (PTP116A100, ST_SysGetPhrDI)

(* Read Inp 0 to 7 from module with address 0. *)

```

FBData.Address:=0;
FBData.Mode:=DI_8_LL;
FBData(); (* Execute FB *)
InpValue:=FBData.Value; (* Digital input value *)

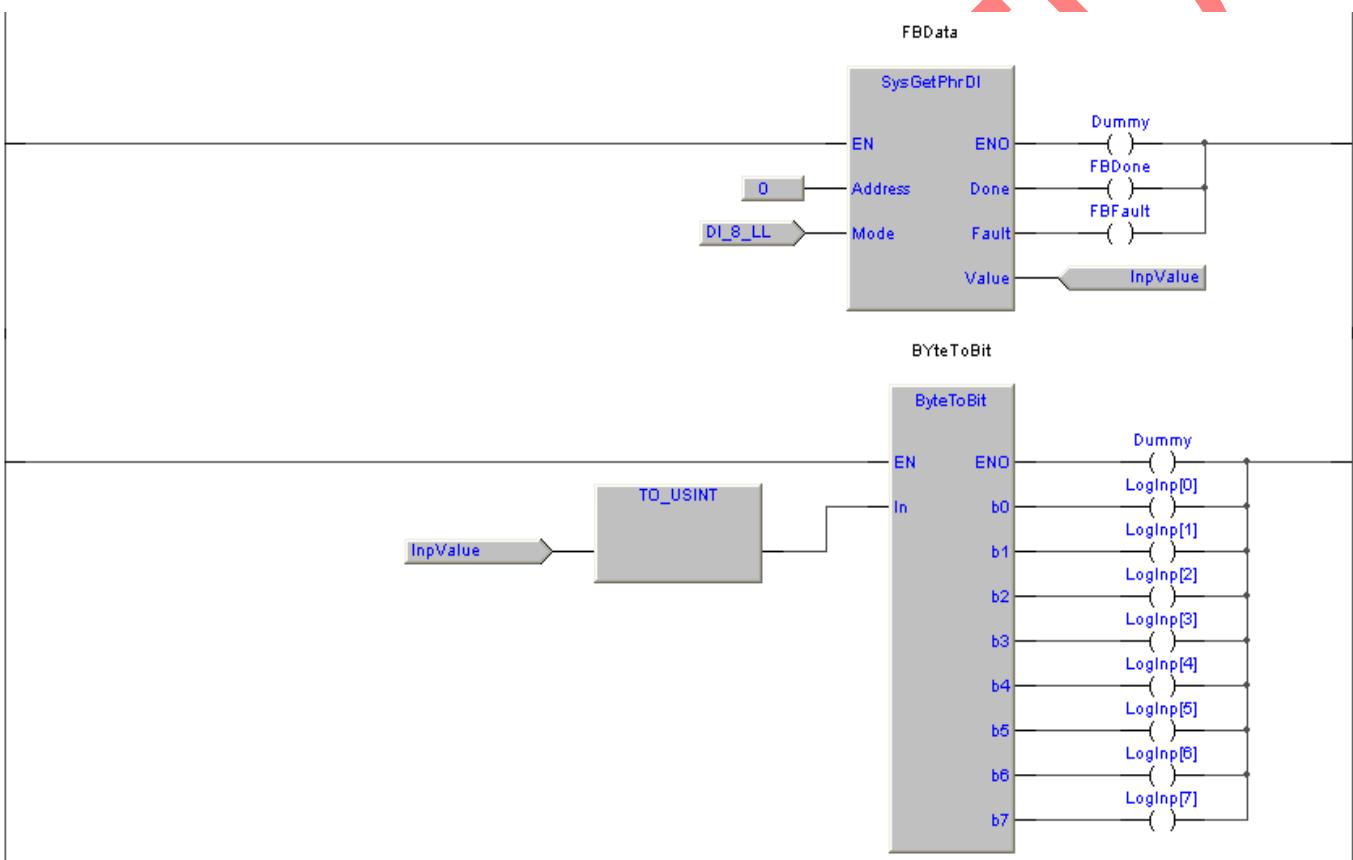
```

This example is an evolution of the previous example. Using the **ByteToBit** function block, the status of the 8 low inputs (from Inp 0 to Inp Inp 7) of the module with address 0, is copied on an array of BOOL.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	FBData	SysGetPhrDI	Auto	No	0	..	FB SysGetPhrDI data
2	InpValue	UDINT	Auto	No	0	..	Digital input value
3	Dummy	BOOL	Auto	No	FALSE	..	Dummy variable
4	FBDone	BOOL	Auto	No	FALSE	..	FB done
5	FBFault	BOOL	Auto	No	FALSE	..	FB fault
6	BYteToBit	ByteToBit	Auto	No	0	..	FB ByteToBit
7	LogInp	BOOL	Auto	[0..7]	8(0)	..	Logic inputs

LD example (PTP119A000, LD LogicInputAcquisition)



7.12.3 SysSetPhrDO, set peripheral digital output

Type	Library	Version
FB	Embedded	XTarget 11.0

This function block performs the setting of digital outputs on the peripheral modules addressed with **Address** and using the **Mode** defined.

To not disturb the logic outputs management executed by the process image, in **Mask** it's possible to define on what output you want to work. In practice, the FB will work only on output whose corresponding bit of the mask is active.

To set the digital outputs on the CPU module, you must define 255 as **Address** and DO_8_LL as **Mode**.



Address (USINT)	You must specify the module address to which to set the digital outputs (range 0 to 255). A value of 0 indicates the first extension module, 1 for the second and so on. The address 255 indicates the CPU module.
Mode (USINT)	You must specify the mode of setting of the digital outputs, refer to Digital output mode .
Value (UDINT)	Set the value to be transferred to the digital outputs.
Mask (UDINT)	Digital output mask.
Done (BOOL)	Data set. Is activated for a loop at the end of setting of the digital outputs.
Fault (BOOL)	Error. It is activated when an error occurs in the settings sequence.

Error codes

If an error occurs, the **Fault** output is activated and [SysGetLastError](#) can detect the error code.

- 9984050 Function block allocation error.
- 9984060 Relocatable memory space full. You can not run the FB.
- 9984070 Function block version error.
- 9984100 The module addressed with **Address** is not present.
- 9984110~8 The acquisition **Mode** is not correct.
- 9984200~6 Error during execution of function block.

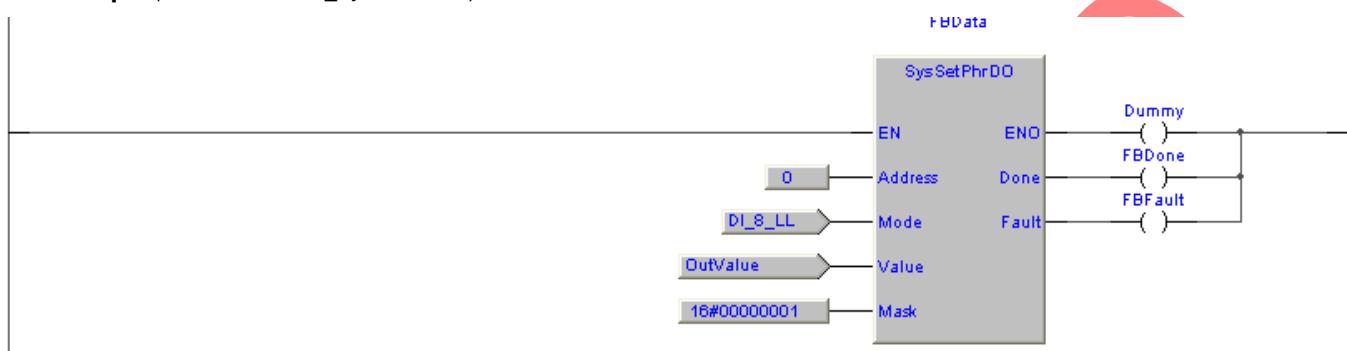
Examples

The value of the **OutValue** variable is transferred to the Out 0 of the module with address 0.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	FBData	SysSetPhrDO	Auto	No	0	..	FB SysSetPhrDO data
2	OutValue	UDINT	Auto	No	0	..	Digital output value
3	Dummy	BOOL	Auto	No	FALSE	..	Dummy variable
4	FBDone	BOOL	Auto	No	FALSE	..	FB done
5	FBFault	BOOL	Auto	No	FALSE	..	FB fault

LD example (PTP116A400, LD_SysSetPhrDO)



IL example (PTP116A400, IL_SysSetPhrDO)

(* Manage digital outputs Out 0 to Out 7 on module with address 0. *)

```

LD 0
    ST FBData.Address
    LD DO_8_LL
    ST FBData.Mode
    LD OutValue
    LD FBData.Value
    LD 16#00000001
    LD FBData.Mask
    CAL FBData

```

ST example (PTP116A400, ST_SysSetPhrDO)

(* Manage digital outputs Out 0 to Out 7 on module with address 0. *)

```

FBData.Address:=0;
FBData.Mode:=DO_8_LL;
FBData.Value:=OutValue; (* Digital output value *)
FBData.Mask:=16#00000001; (* Output mask *)
FBData(); (* Execute FB *)

```

7.12.4 SysGetAnInp, get analog input

Type	Library	Version
FB	Embedded	XTarget 07.0

This function block performs the acquisition of the analog input from the acquisition module. The function block operates according to various methods of acquiring the analog module to which it refers.

To acquire the analog inputs on the CPU module, you must define 255 as **Address** and AD_VOLT_0_10_COMMON as **Mode**.



Address (USINT)	You must specify the module address to which to acquire the analog inputs (range 0 to 255). A value of 0 indicates the first extension module, 1 for the second and so on. The address 255 indicates the CPU module.
Channel (USINT)	You must specify the channel address on the extension module (range from 0x00 to 0x0F). If you set a channel address that is not present, execution stops and the Fault bit is set.
Mode (USINT)	You must specify the mode of analog acquisition, refer to Analog to digital mode .
Done (BOOL)	Analog data acquired. Activated if the analog inputs acquisition, has terminated.
Fault (BOOL)	Acquisition error. It is activated when an error occurs in the sequence of acquisition.
Value (REAL)	Return the acquisition value expressed in units defined by mode of acquisition. It could be a NaN (Not a Number) to indicate a problem in the acquisition, typically sensor failure.

Error codes

If an error occurs, the **Fault** output is activated and [SysGetLastError](#) can detect the error code.

- 9983050 Function block allocation error.
- 9983060 Relocatable memory space full. You can not run the FB.
- 9983070 Function block version error.
- 9983080 Impossible to initialize module.
- 9983100 The module addressed with **Address** is not present.
- 9983110~1 The addressed module does not support the analog input command.
- 9983150 The returned value is not correct.
- 9983200 The selected **Mode** is not supported by module.
- 9983210 Error during analog acquisition.
- 9983300 The selected **Channel** is not supported by module.

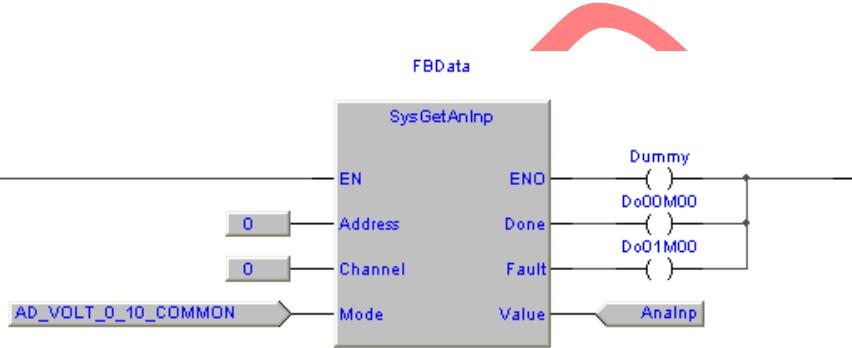
Examples

Acquisition from analog channel 0 of module 0 using 0-10 volts mode is executed. If the data is valid, the **Do00M00** digital output is activated. If conversion error, the **Do01M00** digital output is activated. The analogue data acquired in the range from 0,000 to 9,999 is transferred into the variable **AnaInp**.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	FBData	SysGetAnInp	Auto	No	0	..	FB SysGetPhrDI data
2	Dummy	BOOL	Auto	No	FALSE	..	Dummy variable
3	AnaInp	REAL	Auto	No	0	..	Analog value (Volts)

LD example (PTP116A100, LD_SysGetAnInp)



IL example (PTP116A100, IL_SysGetAnInp)

```

(* Acquires analog input 0 from module. *)

LD 0
ST FBData.Address (* Set module address *)

LD 0
ST FBData.Channel (* Set channel *)

LD AD_VOLT_0_10_COMMON
ST FBData.Mode (* Set acquisition mode *)

CAL FBData (* Call the SysGetAnInp function block *)

LD FBData.Done
ST Do00M00 (* The output is active if data is acquired *)

LD FBData.Fault
ST Do01M00 (* The output is active if execution fault *)

LD FBData.Value
ST AnaInp (* Store the acquired value *)
  
```

ST example (PTP116A100, ST_SysGetAnInp)

```

(* Acquires analog input 0 from module. *)

FBData(Address:=0, Channel:=0, Mode:=AD_VOLT_0_10_COMMON); (* Call the SysGetAnInp FB *)

Do00M00:=FBData.Done; (* The output is active if data is acquired *)
Do01M00:=FBData.Fault; (* The output is active if execution fault *)
VarReal:=FBData.Value; (* Store the acquired value *)
  
```

7.12.5 SysSetAnOut, set analog output

Type	Library	Version
FB	Embedded	XTarget 07.0

This function block performs the set of a value on the analog output module. The function block manages a variety of ways depending on the analog module to which it refers.



Address (USINT)	You must specify the address of the module to which to set the analog value (range from 0x00 to 0x0F). The value 0x00 indicates the first extension module, 0x01 the second and so on.
Channel (USINT)	You must specify the channel address on the extension module (range from 0x00 to 0x0F). If you set a channel address that is not present, execution stops and the Fault bit is set.
Mode (USINT)	You must specify the mode of analog output, refer to Digital to analog mode .
Value (REAL)	You must specify the output value in the unity defined by Mode .
Done (BOOL)	Data set. Is activated for a loop at the end of setting of the analog output.
Fault (BOOL)	Error. It is activated when an error occurs in the settings sequence.

Error codes

If an error occurs, the **Fault** output is activated and [SysGetLastError](#) can detect the error code.

- 9982050 Function block allocation error.
- 9982060 Relocatable memory space full. You can not run the FB.
- 9982070 Function block version error.
- 9982080 Impossible to init module.
- 9982100 The module addressed with **Address** is not present.
- 9982110~1 The addressed module does not support the analog output command.
- 9982150 The value to set is not correct.
- 9982200 The selected **Mode** is not supported by module.
- 9982210 Error during analog output.

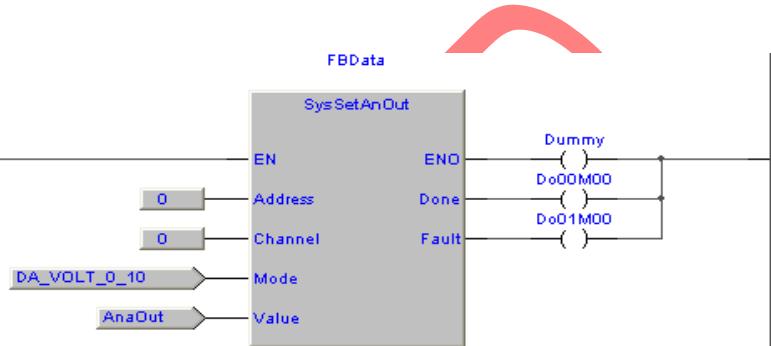
Examples

The analog output on channel 0 of module 0 using 0-10 volts mode, is executed. If the data is valid, the **Do00M00** digital output is activated. If conversion error, the **Do01M00** digital output is activated. The data to set to analog output in the range from 0,000 to 9,999 is present in the **AnaOut** variable.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	FBData	SysSetAnOut	Auto	No	0	..	FB SysSetAnOut data
2	Dummy	BOOL	Auto	No	FALSE	..	Dummy variable
3	AnaOut	REAL	Auto	No	0	..	Analog value (Volts)

LD example (PTP116A100, LD_SysSetAnOut)



IL example (PTP116A100, IL_SysSetAnOut)

```

(* Manage analog output 0 on module 0. *)

LD 0
ST FBData.Address (* Set module address *)

LD 0
ST FBData.Channel (* Set channel *)

LD DA_VOLT_0_10
ST FBData.Mode (* Set management mode *)

LD AnaOut
ST FBData.Value (* Store the output value *)

CAL FBData (* Call the SysSetAnOut function block *)

LD FBData.Done
ST Do00M00 (* The output is active if data is set *)

LD FBData.Fault
ST Do01M00 (* The output is active if execution fault *)

```

ST example (PTP116A100, ST_SysSetAnOut)

```

(* Manage analog output 0 on module 0. *)

FBData.Value:=AnaOut; (* Store the ouput value *)
FBData(Address:=0, Channel:=0, Mode:=DA_VOLT_0_10); (* Call the SysSetAnOut function block *)

Do00M00:=FBData.Done; (* The output is active if data is set *)
Do01M00:=FBData.Fault; (* The output is active if execution fault *)

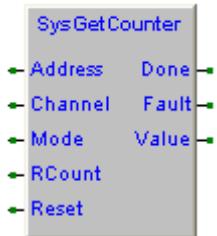
```

7.12.6 SysGetCounter, get counter

Type	Library	Version
FB	Embedded	XTarget 07.0

This function block reads a counter. The function block can be used to capture the value of the counter on the CPU module and the SlimLine modules that support the counter function.

It is possible to manage the reset and the reverse of the count value. According to the definition of **Mode**, you can manage count on rising edge, falling edge or both edge of the input clock. If the module that manages the counter support it, you can also define hardware commands (digital inputs) to reset counter and to reverse counting.



Address (USINT) You must specify the module address from which to get counter value (range from 0 to 255). The value 0 indicates the first extension module, 1 for the second and so on. The address 255 indicates the CPU module.

If you set an address of a module that is not present, execution stops and the **Fault** bit is set.

Channel (USINT) You must specify the channel address on the module (range from 0 to 15).

If you set a channel address that is not present, execution stops and the **Fault** bit is set.

Mode (UDINT) Acquisition mode. It is a 32 bit value organized as below.

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

Mode	Input	Mode	Input	Mode	Input
------	-------	------	-------	------	-------

Reverse Reset Clock

Clock	Input	Defines the digital input to use as clock
	Mode	0: Count on rising edge 1: Count on falling edge 2: Count on both edge
Reset	Input	Defines the digital input to use as reset
	Mode	0: Not used 1: Reset counter if digital input is active 2: Reset counter if digital input is not active
Reverse	Input	Defines the digital input to use to reverse counting
	Mode	0: Not used 1: Reverse count if digital input is active 2: Reverse count if digital input is not active

To calculate the Mode value, you must apply this formula:

$((Reverse\ mode)*2097152)+((Reverse\ input)*65536)+((Reset\ mode)*8192)+((Reset\ input)*256)+((Clock\ mode)*32)+(Clock\ input)$

If an incorrect value is set, execution stops and the **Fault** bit is set.

RCount (BOOL) Reverse counting. If active, the **Value** is decreased every **Clock**; if deactivate, it is increased every **Clock**.

Reset (BOOL) When active, the **Value** is reset.

Done (BOOL) Counter value acquired. Active for a program loop at the end of acquisition.

Fault (BOOL) Acquisition error. It is activated when an error occurs in the sequence of acquisition.

Value (UDINT) Counter value.

Error codes

If an error occurs, the **Fault** output is activated and [**SysGetLastError**](#) can detect the error code.

- 9981050 Function block allocation error.
- 9981060 Relocatable memory space full. You can not run the FB.
- 9981070 Function block version error.
- 9981080 Impossible to initialize module.
- 9981100 The module addressed with **Address** is not present.
- 9981110 The defined **Channel** is not supported.
- 9981200~4 The defined **Mode** is not supported.
- 9981300~2 Error during reading of counter value from module.
- 9981990 Not yet Implemented in the simulator.

Examples

It is acquired the counter value from the SlimLine CPU module. Counting is performed on both sides of the clock input. The count value is transferred into the **Value** variable. At the end of conversion, the **Do01M00** digital output is activated. If conversion error, the **Do02M00** digital output is activated.

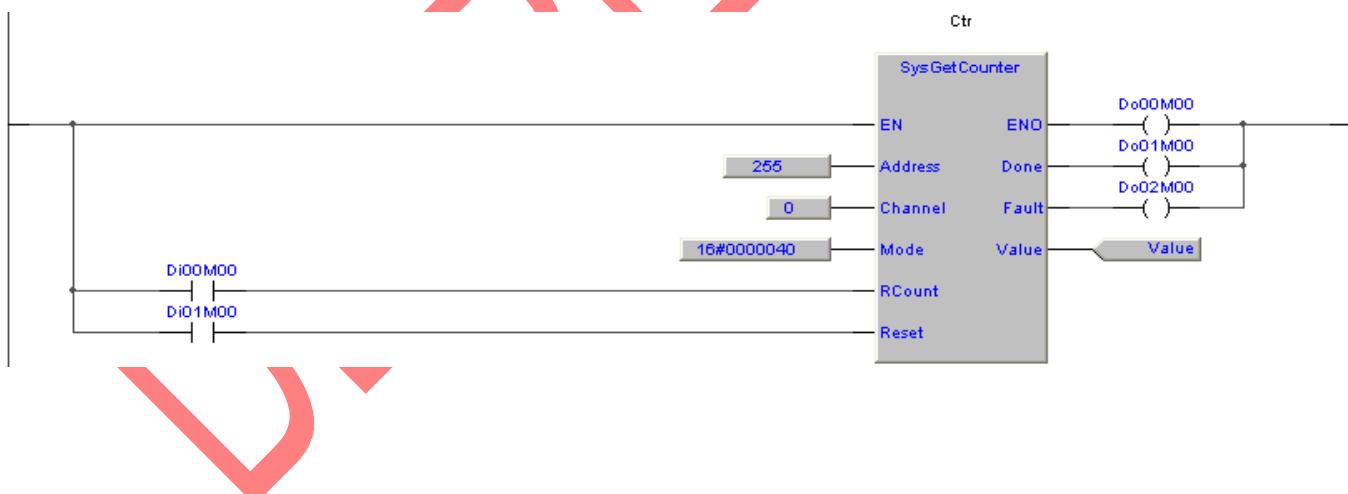
Activating the **Di00M00** input, there is the reverse of the count. At each edge of the clock input, the count **Value** decreases.

Activating the **Di01M00** input, the count **Value** is reset.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Value	UDINT	Auto	No	0	..	Counter value
2	Ctr	SysGetCounter	Auto	No	0	..	SysGetCounter FB data

LD example

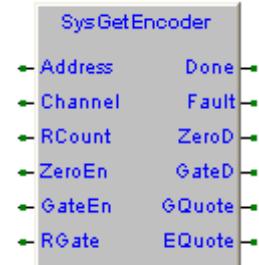


7.12.7 SysGetEncoder, get encoder input

Type	Library	Version
FB	Embedded	XTarget 07.0

This function block reads an encoder channel. The function block can only be used on systems that have extension modules that support incremental encoders.

It is possible to manage the zero mark and to acquire quotes values on a gate signal.



Address (USINT)	You must specify the module address from which to get encoder value (range from 0 to 15). The value 0 indicates the first extension module, 1 for the second and so on. The address 255 indicates the CPU module. If you set an address of a module that is not present, execution stops and the Fault bit is set.
Channel (USINT)	You must specify the channel address on the module (range from 0 to 15). If you set a channel address that is not present, execution stops and the Fault bit is set.
RCount (BOOL)	Reverse counting. Activating this input, it is inverted the EQuote increment related to the direction of rotation of the encoder.
ZeroEn (BOOL)	By activating this input, the EQuote will be reset at the passage of the encoder zero mark.
GateEn (BOOL)	By activating this input, on the rising or falling edge (depending from Rgate) of Gate input, the EQuote value will be copied in the GQuote value.
RGate (BOOL)	Activating this input, the falling edge of Gate input will be used.
Done (BOOL)	Encoder value acquired. Active for a program loop at the end of acquisition.
Fault (BOOL)	Acquisition error. It is activated when an error occurs in the sequence of acquisition.
ZeroD (BOOL)	Encoder zero mark acquired. It is activated when the zero mark input is detected. It is reset deactivating the ZeroEn input.
GateD (BOOL)	Encoder Gate acquired. Active for a program loop when the Gate signal is detected.
GQuote (UINT)	Gate quote. This is the EQuote value stored on the edge selected for Gate signal.
EQuote (UINT)	Encoder quote. This is the encoder quote value. When it reaches the minimum or maximum value, there is the roll over.

Error codes

If an error occurs, the **Fault** output is activated and [SysGetLastError](#) can detect the error code.

- 9980050 Function block allocation error.
- 9980060 Relocatable memory space full. You can not run the FB.
- 9980070 Function block version error.
- 9980100 The module addressed with **Address** is not present.
- 9980110~2 The addressed module doesn't support the encoder commands.
- 9980200 The defined **Mode** is not supported by module.
- 9980210~2 Error during reading of encoder value from module.
- 9980990 Not yet Implemented in the simulator.

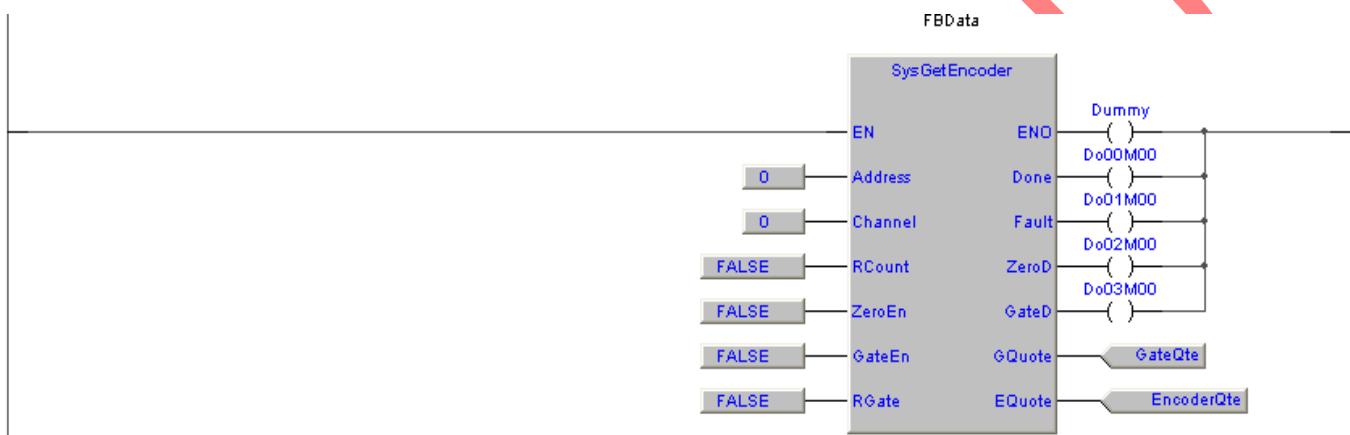
Examples

It is acquired the encoder value from input channel 0 of module 0. The encoder value is transferred into the **EncoderQte** variable. At the end of acquisition, the **Do00M00** digital output is activated. If conversion error, the **Do01M00** digital output is activated. If conversion error, the **Do02M00** digital output is activated. If conversion error, the **Do03M00** digital output is activated.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	FBData	SysGetEncoder	Auto	No	0	..	FB SysGetEncoder data
2	Dummy	BOOL	Auto	No	FALSE	..	Dummy variable
3	GateQte	UINT	Auto	No	0	..	Gate quote
4	EncoderQte	UINT	Auto	No	0	..	Encoder quote

LD example (PTP116A100, LD_SysGetEncoder)



IL example (PTP116A100, IL_SysGetEncoder)

(* Acquires encoder 0 from module. *)

```

LD 0
ST FBData.Address (* Set module address *)
LD 0
ST FBData.Channel (* Set channel *)
LD FALSE
ST FBData.RCount (* Reverse counting *)
ST FBData.GateEn (* Gate enable *)
ST FBData.RGate (* Reverse gate *)
CAL FBData (* Call the SysGetEncoder function block *)
LD FBData.Done
ST Do00M00 (* The output is active if data is acquired *)
LD FBData.Fault
ST Do01M00 (* The output is active if execution fault *)
LD FBData.GQuote
ST GateQte (* Gate quote *)
LD FBData.EQuote
ST EncoderQte (* Encoder quote *)

```

7.12.8 SysSetPWMOut, set PWM output

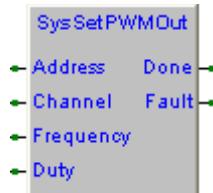
Type	Library	Version
FB	Embedded	XTarget 07.0

This function block sets the value of frequency and duty cycle of the PWM generator selected by **Address** e **Channel**. The value range of frequency and duty cycle depends on the card used.

Setting the value of **Frequency** to 0 it disables the PWM and the output is forced based on the value of **Duty**.

Duty <50: The output is set to FALSE.

Duty >= 50: The output is set to TRUE.



- Address** (USINT) You must specify the module address from which to get encoder value (range from 0 to 15). The value 0 indicates the first extension module, 1 for the second and so on. The address 255 indicates the CPU module.
If you set an address of a module that is not present, execution stops and the **Fault** bit is set.
- Channel** (USINT) You must specify the channel address on the module (range from 0 to 15).
If you set a channel address that is not present, execution stops and the **Fault** bit is set.
- Frequency** (REAL) Output frequency value (For range refer to the hardware manual of the card used). The value is expressed in **Hz**.
- Duty** (REAL) Duty cycle output value (For range refer to the hardware manual of the card used). The value is expressed in %.
- Done** (BOOL) PWM generator properly set.
- Fault** (BOOL) Execution error.

Error codes

If an error occurs, the **Fault** output is activated and [SysGetLastError](#) can detect the error code.

- 9951050 Function block allocation error.
- 9951070 Function block version error.
- 9951100 The module addressed with **Address** is not present.
- 9951110 Channel number not present.
- 9951500 Execution error.
- 9951990 Not yet Implemented in the simulator.

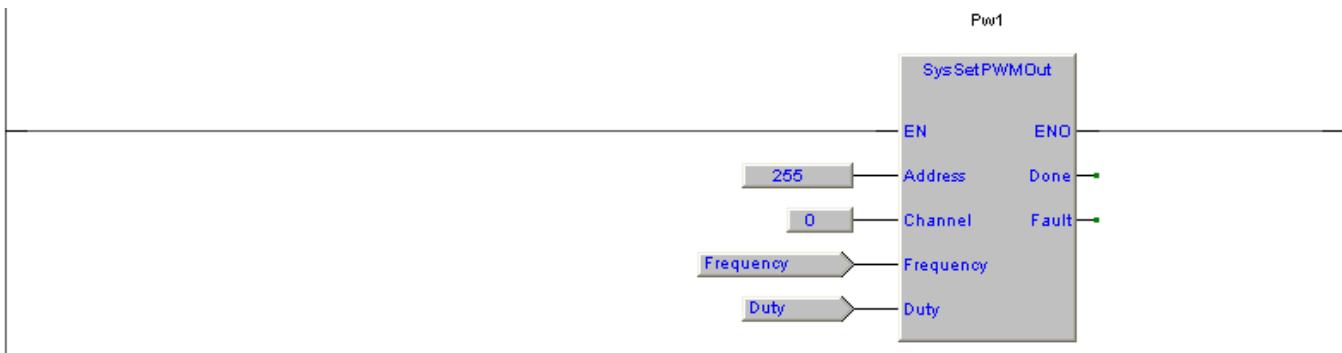
Esempi

Is set output Out 0 of the CPU module to generate a PWM signal at 100 Hz with 50% duty cycle.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Pw1	SysSetPWMOut	Auto	No	0	..	FB data
2	Frequency	REAL	Auto	No	100	..	Frequency set (Hz)
3	Duty	REAL	Auto	No	50	..	DutyCycle set (%)

LD example



DEPRECATA

7.12.9 SysPhrVRd, read variable from peripheral module

Type	Library	Version
Function	Embedded	XTarget 07.0

This function reads a variable from the extension module.

You must define the **Module** address, the **RdAdd** address of the variable to read, the **VarType** type of variable and the **VarAdd** buffer address where to transfer the read value.



Function parameters:

- Module** (USINT) You must specify the module address from which to get value (range from 0 to 15). The value 0 indicates the first extension module, 1 for the second and so on.
- RdAdd** (UINT) Address of variable to read as allocated on the extension module.
- VarType** (USINT) Variable type as defined in the [variable types definition](#) table.
- VarAdd** (UDINT) Address of variable where to copy the read value.

Return value:

- (BOOL) **FALSE**: Execution error. **TRUE**: Execution ok.

Error codes

If an error occurs, the function return **FALSE** and [**SysGetLastError**](#) can detect the error code.

- 9989100 The module addressed with **Module** is not present.
- 9989110 The variable type **VarType** is not correct.
- 9989200 Error during read of variable value from module.
- 9989990 Not yet Implemented in the simulator.

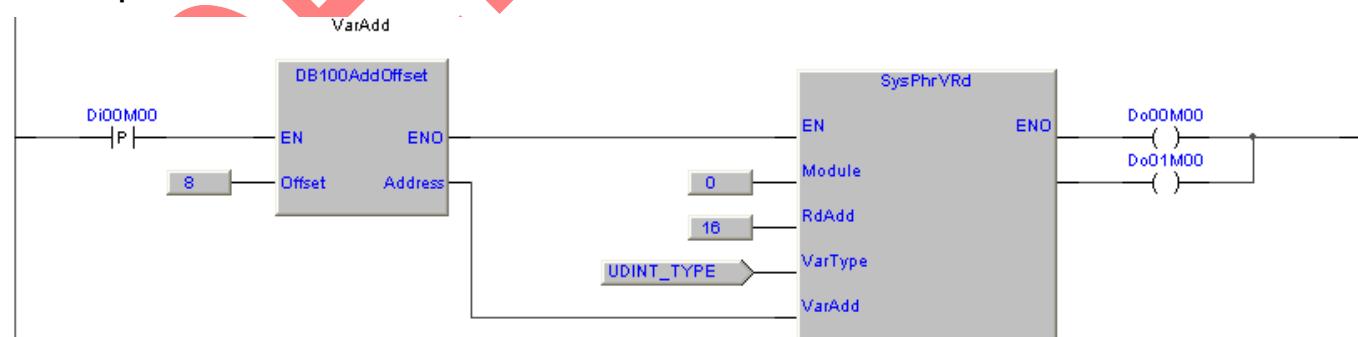
Examples

By activating the **Di00M00** input, the read of **UDINT** variable at address **16** of the extension module 0 is executed. The value of the variable is copied to **DB100** at offset **8**.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	VarAdd	DB100AddOffset	Auto	No	0	..	Variable address calculation

LD example



7.12.10 SysPhrVWr, write variable to peripheral module

Type	Library	Version
Function	Embedded	XTarget 07.0

This function write a variable to the extension module.

You must define the **Module** address, the **WrAdd** address of the variable to write, the **VarType** type of variable and the **VarAdd** buffer address where there is the value to write.



Function parameters:

- Module** (USINT) You must specify the module address to which to write value (range from 0 to 15). The value 0 indicates the first extension module, 1 for the second and so on.
- WrAdd** (UINT) Address of variable to write as allocated on the extension module.
- VarType** (USINT) Variable type as defined in the [variable types definition](#) table.
- VarAdd** (UDINT) Address of variable where there is the value to write

Return value:

(BOOL) **FALSE**: Execution error. **TRUE**: Execution ok.

Error codes

If an error occurs, the function return **FALSE** and [SysGetLastError](#) can detect the error code.

9988100 The module addressed with **Module** is not present.

9988110 The variable type **VarType** is not correct.

9988200 Error during write of variable value to module.

9988990 Not yet Implemented in the simulator.

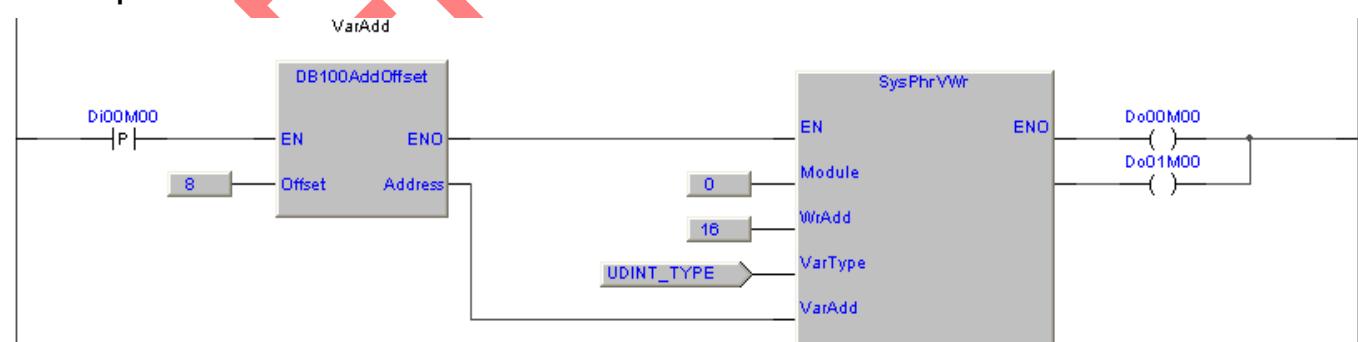
Examples

By activating the **Di00M00** input, the write of **UDINT** variable at address **16** on the extension module 0 is executed. The value to write is in **DB100** at offset **8**.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	VarAdd	DB100AddOffset	Auto	No	0	..	Variable address calculation

LD example



7.12.11 SysI2CWrRd, writes/reads on I2C extension bus

Type	Library	Version
Function	Embedded	XTarget 07.0

This function manages the I2C extension bus. By using this function it's possible to manage any component connected to the I2C extension bus.

Be careful, the I2C bus is used to access to extension modules to avoid to slow down their access we suggest to manage I2C commands not longer than 4 bytes in writing and reading. If needed to manage longer commands it is advisable to break the command into multiple commands.



Function parameters:

Address (USINT) I2C address out of range, range is from 16#00 to 16#7F.

WrBytes (USINT) Number of bytes to write. 0 if only bus read.

WrBuffer (@USINT) Write data buffer address. NULL if only bus read.

RdBytes (USINT) Number of bytes to read. 0 if only bus write.

RdBuffer (@USINT) Read data buffer address. NULL if only bus write.

Return value:

(BOOL) **TRUE**: Command executed.

Error codes

If an error occurs the function returns **FALSE** and **SysGetLastError** can detect the error code.

9953100 I2C address out of range (Value greater than 16#7F).

9953105 I2C address used by an extension module.

9953990 Not yet Implemented in the simulator.

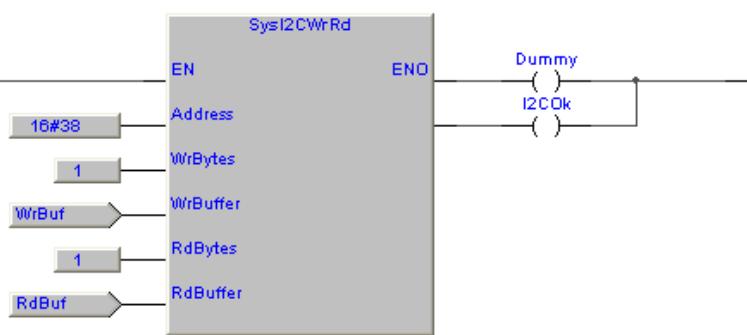
Examples

In this example a simple program that reads and writes a PCF9670 I/O expander on I2C bus. The chip has address 16#38.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Dummy	BOOL	Auto	No	FALSE	..	Dummy variable
2	I2COK	BOOL	Auto	No	FALSE	..	I2C Ok
3	RdBuf	BYTE	Auto	No	0	..	Read data buffer
4	WrBuf	BYTE	Auto	No	0	..	Write data buffer

LD example



7.12.12 StrainGaugeAcq, strain gauge acquisition

Type	Library	Version
FB	eLLabUtyLib_C030	SFR054B220

This function block manages the strain gauge acquisition it uses the **SysGetAnInp** FB to acquire the analog input that reads the strain gauge. In **Address** and **Channel** must be defined the analog input acquisition module parameters. In **Mode** must be defined the appropriate strain gauges acquisition mode.

Before to operate it's important to calibrate the acquisition (Calibration values are stored in **OfsCalibration** and **FSCalibration** variables). To calibrate the offset connect the strain gauge and with no weight applied activate for a loop **DoOfsCalibration**. For the full scale calibration apply a weight as possible next to the full scale value to the strain gauge. The value of the applied weight must be defined in **FSReference**, after a period of weight reading stabilization to activate for a loop **DoFSCalibration**.

Mode must be set higher. But this formula must be respected: **(SGaugeRatio x PowS. x Mode) < PowS.** Where **PowS.** is the strain gauge power supply. Example: strain gauge with **SGaugeRatio** = 2mV/V and power supply 5V. With Mode = AD_VIN_VREF_G_128 will be $(2\text{mV/V} * 5\text{V} * 128) = 1280\text{mV}$ so under the 5V power supply and so ok.

In **FilterCf** è possibile definire un coefficiente di filtro sulla acquisizione del peso, più il valore è grande e più il valore di peso è filtrato. Con 0, non c'è filtro lasciando la possibilità di inserire un proprio filtro personalizzato esterno.

In **FilterCf** you can define a filter coefficient on weight acquisition, more the value is great and more the weight value is filtered. With 0, there is no filter so it is possible to insert an external custom filter.



Enable (BOOL)	FB enable.
DoOfsCalibration (BOOL)	Offset calibration command.
DoFSCalibration (BOOL)	Full scale calibration command.
Address (USINT)	You must specify the module address to which acquire the strain gauge (range 0 to 255). A value of 0 indicates the first extension module, 1 for the second and so on.
Channel (USINT)	You must specify the channel address on the extension module (range from 0x00 to 0x0F). If you set a channel address that is not present, execution stops and the Fault bit is set.
Mode (USINT)	Input acquisition mode, analog to digital mode .
FilterCf (REAL)	Acquisition filter (Nr)
FSReference (REAL)	Reference value used to calibrate the strain gauge full scale (Nr)
SGaugeFullScale (REAL)	Strain gauge full scale (Strain gauge characteristic) (Nr)
SGaugeRatio (REAL)	Strain gauge mv/V ratio (Strain gauge characteristic) (Nr)
OfsCalibration (@REAL)	Offset calibration storage variable address (Variable must be RETAIN)
FSCalibration (@REAL)	Full scale calibration storage variable address (Variable must be RETAIN)
Enabled (BOOL)	FB enabled.
Done (BOOL)	Strain gauge value acquired.
Fault (BOOL)	Acquisition fault.
Value (REAL)	Acquisition value it has the same measure unit of the FSReference and SGaugeFullScale .

Error codes

If an error occurs the function returns **FALSE** and **SysGetLastError** can detect the error code.

10045100 **Mode** value not correct.

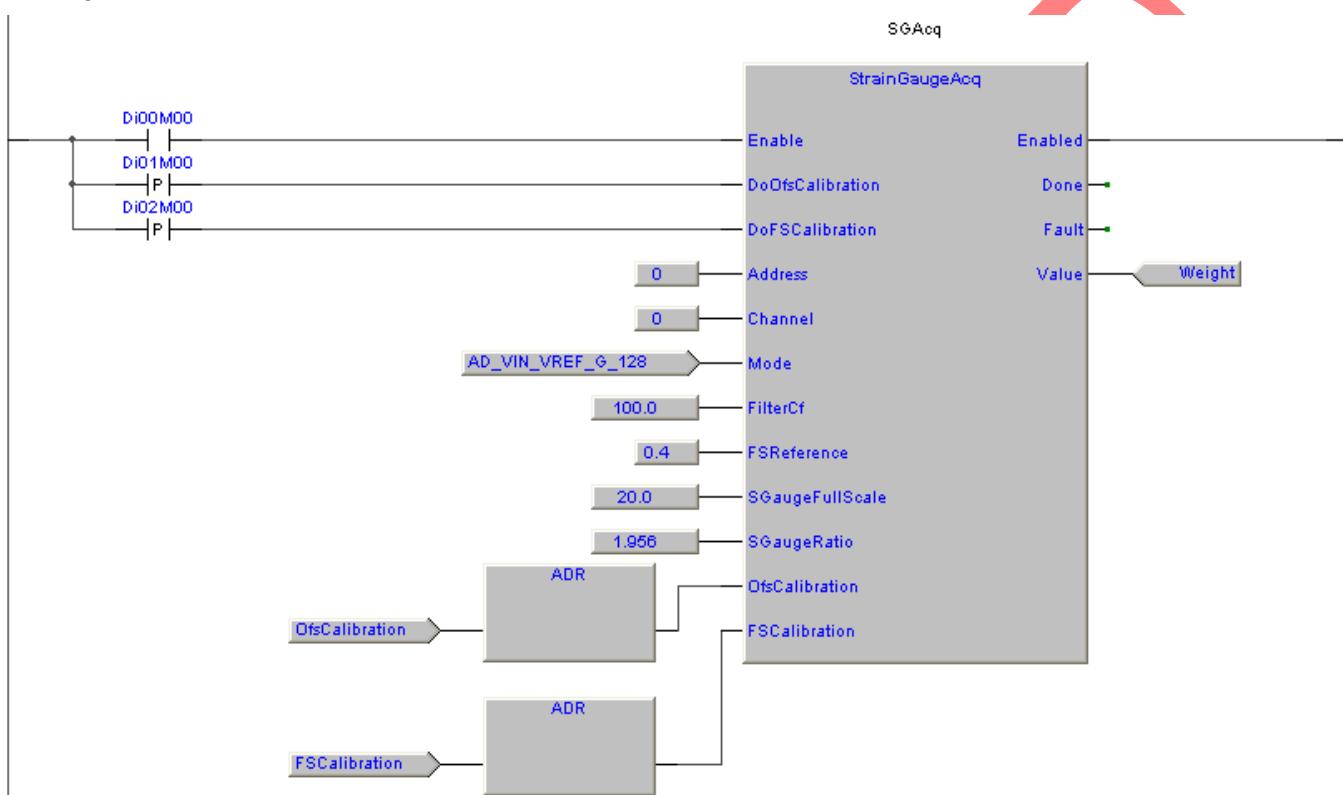
Example

The program acquires a load cell with 1.956 m/V and full scale of 20 Kg. The cell is calibrated with a reference weight of 400 grams. **Di00M00** enables the acquisition, inputs **Di01M00** and **Di02M00** allow to calibrate the cell (Caution **OfsCalibration** and **FSCalibration** variables must be defined RETAIN). Weight is returned in the variable **Weight**. Having defined **SGaugeFullScale** in Kg **Value** is also expressed in Kg.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Weight	REAL	Auto	No	0	..	Weight (Kg)
2	SGAcq	StrainGaugeAcq	Auto	No	0	..	Strain gauge acquisition

LD example

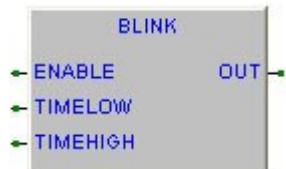


7.13 Functions and FBs of general utility

7.13.1 BLINK, blink command

Type	Library
FB	eLLabUtyLib_C030

This function block operates a blinking output with definable cycle time. By activating the **ENABLE** input, the **OUT** output blinks with defined high and low cycle times.



- ENABLE** (BOOL) Function block enable. Activating it, the blinking of OUT output is handled. Deactivating it the OUT output is reset.
- TIMELOW** (UDINT) Defines the time in which the OUT output is in low state. It is expressed in mS.
- TIMEHIGH** (UDINT) Defines the time in which the OUT output is in high state. It is expressed in mS.
- OUT** (BOOL) Blinking output.

Examples

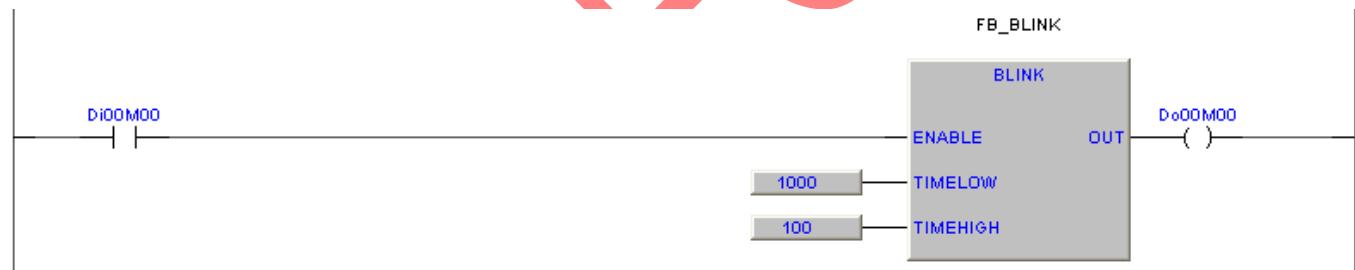
Is set a blink with 100 ms in high state and 1000 ms in low state. Activating the **Di00M00** digital input, the **Do00M00** digital output blinks with the time defined.

Deactivating the **Di00M00** digital input, the **Do00M00** digital output resets immediately.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	FBeBLINK	BLINK	Auto	No	0	..	eBLINK (Blink out function block)

LD example (PTP14A100, LD_BLINK)



IL example (PTP14A100, IL_BLINK)

```

CAL FB_BLINK (* Call the BLINK function block *)
LD Di00M00
ST FB_BLINK.ENABLE (* Transfer the digital input to enable input *)
LD 1000
ST FB_BLINK.TIMELOW (* Set the time low *)
LD 100
ST FB_BLINK.TIMEHIGH (* Set the time high *)
LD FB_BLINK.OUT
ST Do00M00 (* Copy FB output to logic output *)

```

ST example (PTP14A100, ST_BLINK)

```

FB_BLINK(TIMELOW:=1000, TIMEHIGH:=100); (* Call the BLINK function block *)
FB_BLINK.ENABLE:=Di00M00; (* Transfer the digital input to FB enable *)
Do00M00:=FB_BLINK.OUT; (* Transfer the FB output to digital output *)

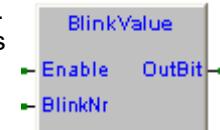
```

7.13.2 BlinkValue, blink out value

Type	Library
FB	eLLabUtyLib_C030

This function block executes a blinking output with the possibility to define the number of blinks. Activating the **Enable** input, and defining the number of blinks in **BlinkNr**, the **OutBit** output blinks with the number of blinks defined.

The number of blinks is defined in the tens and units. The value of tens is shown with a slow blink (1 sec), while the number of units is shown by blinking fast (250 ms). There is a pause of 3 sec between blinking sequences.



Enable (BOOL) Enable function block. Activating it, the **OutBit** output is managed. Deactivating it, the output is reset.

BlinkNr (USINT) Defines the number of blink for the **OutBit** output. By defining time 0, the output is disabled.

OutBit (BOOL) Blinking output status.

Examples

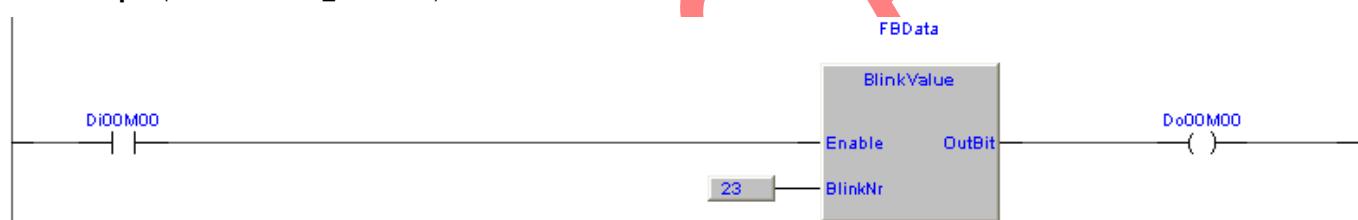
Activating the **Di00M00** digital input, the **Do00M00** digital output blinks with 2 blinks slow (1 sec), 3 blinks fast (250 ms) and a pause for 3 Sec.

Deactivating the **Di00M00** digital input, the **Do00M00** digital output resets immediately.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	FBData	BlinkValue	Auto	No	0	..	FB BlinkValue data

LD example (PTP14A100, LD_BlinkValue)



IL example (PTP14A100, IL_BlinkValue)

```

CAL FBData (* Call the "BlinkValue" function block *)

LD Di00M00
ST FBData.Enable (* Transfer the digital input to enable input *)

LD 23
ST FBData.BlinkNr (* Set the number of blink *)

LD FBData.OutBit
ST Do00M00 (* Copy FB output to logic output *)

```

ST example (Ptp114a100)

```

FBData(BlinkNr:=23); (* Call the BLINK function block *)

FBData.Enable:=Di00M00; (* Transfer the digital input to FB enable *)
Do00M00:=FBData.OutBit; (* Transfer the FB output to digital output *)

```

7.13.3 ModbusMaster, modbus master

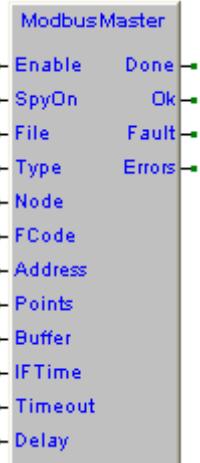
Type	Library
FB	eLLabUtyLib_C030

This function performs the management of Modbus RTU, Ascii, TCP master protocol (Selectable by **Type**). You can define the **File** I/O terminal on which to communicate. Setting the **Enable**, on the terminal I/O is sent a frame to execute the modbus function defined in the **FCode**. Once the command execution is finished the **Done** output is activated. If the command execution is successful the **Ok** output will be set for a program loop. By disabling **Enable** the **Done** and **Fault** output are reset, to run again the command the **Enable** input must re-enabled.

If **FCode** is a read function, the value of variables starting from **Address** for the number of variables defined by **Points**, is read from the slave system and transferred into the variables addressed by **Buffer**. The **SpyOn** input allows to spy the FB working.

If **FCode** is a write function, the value of variables addressed by **Buffer** for the number of variables defined by **Points**, is written to the slave system starting from **Address** address.

On error executing command or command execution time bigger than the defined **Timeout**, the **Fault** output will be activated for a program loop and the **Errors** counter increased.



Enable (BOOL) Command that enables Modbus command execution. To re-execute the command, disable and then re-enable this input.

SpyOn (BOOL) Active allows to spy the FB working.

File (FILEP) Stream returned from [Sysfopen](#) function.

Type (USINT) Modbus type selector. 0:RTU, 1:Ascii, 2:TCP

Node (USINT) Modbus node number on which to execute command (Range from 0 to 255).

FCode (USINT) Modbus code function to execute (Range from 0 to 255).

Code	Description
16#01	Read coil status (Max 255 coils)
16#02	Read input status (Max 255 inputs)
16#03	Read holding registers (Max 32 registers)
16#04	Read input registers (Max 32 registers)
16#06	Preset single register
16#0F	Force multiple coils (Max 255 coils)
16#10	Preset multiple registers (Max 32 registers)

Address (UINT) Address allocation of variables on slave system. According to the Modbus specifications, the address sent in the frame is (**Address-1**) (Range from 16#0001 to 16#FFFF).

Points (USINT) Number of consecutive variables on which the command operates.

Buffer (@USINT) Address of buffer of read or written data.

IFTIME (UDINT)Time between frame (μ S).

If you use the serial port, this time must be related to baud rate.

Baud rate	Time
300	112000
600	56000
1200	28000
2400	14000
4800	7000
9600	3430

Baud rate	Time
19200	1720
38400	860
57600	573
76800	429
115200	286

On ethernet connection set the minimum time value.

Timeout (UINT)Maximum command execution time expressed in mS. If the command does not end in the defined time, the command is aborted and the **Fault** output is activated.**Delay** (UINT)

Pause time after the execution of the command modbus expressed in mS.

Done (BOOL)

Active at the end of command execution.

Ok (BOOL)

Active for a program loop if command correctly executed.

Fault (BOOL)

Active for a program loop if execution error.

SpyTrg (UDINT)

Trigger for the SpyDataFile FB.

SpyPtr (@USINT)

Pointer to data to be spy.

Errors (UDINT)

Error counter. Increased at every error. If it reach the maximum value, it restarts from 0.

Spy triggerIf **SpyOn** is active the [SysSpyData](#) function is executed this allows to spy the FB operations. There are various levels of triggers.**TFlags** **Description**16#00000001 **Tx**: Modbus command frame sent.16#00000002 **Rx**: Modbus answer frame received.**Error codes**If an error occurs, the **Fault** output is activated and [SysGetLastError](#) can detect the error code.10007010 **File** value not defined.

10007050 Execution timeout.

10007060 Execution error.

10037080 **Type** definition error.10007100 Function code defined in **Function** is not managed.10007120 Wrong **Points** value.

10007200~2 Error transmitting the command frame.

10007500~7 Error receiving the answer frame (Wrong character, wrong length, won CRC).

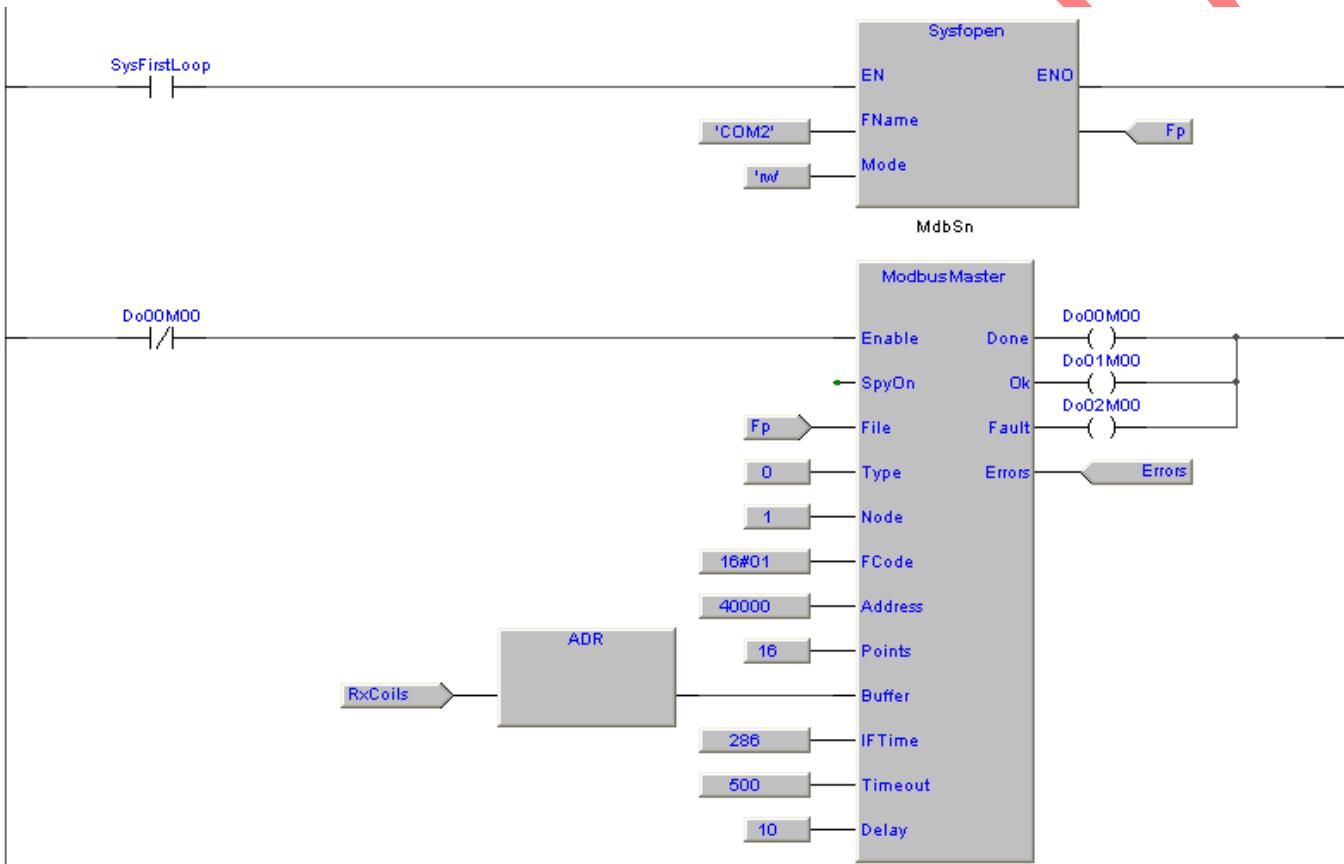
Examples

Here an example of reading from a slave SlimLine. Cyclically is performed a read of 16 coils starting from address 16#01 of Modbus node 1. The value of the coils is transferred to the **RxCoils** array. After the reading will be activated the **Do01M00** logic output for a program loop.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	RxCoils	BOOL	Auto	[0..15]	16(0)	..	Rx coils data buffer
2	Errors	UDINT	Auto	No	0	..	Modbus communication errors
3	Fp	FILEP	Auto	No	0	..	File pointer
4	MdbSn	ModbusMaster	Auto	No	0	..	Modbus master FB

LD example (PTP114A620, LD_ModbusMaster)



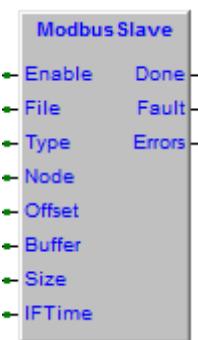
7.13.4 ModbusSlave, modbus slave

Type	Library
FB	eLLabUtyLib_C030

SlimLine systems have got the Modbus slave protocol already implemented in the operating system, so does not need a special function blocks in the user program. This block overrides the management of the operating system and is used in special cases, where you can not use the management implemented in the operating system. For example when you want to allow the access to your own memory area different from DB100.

This function block performs the management of Modbus RTU, Ascii, TCP slave protocol (Selectable by **Type**). It is possible to define the terminal I/O **File** on which to work.

It is possible to define the modbus **Node** and the frame address **Offset**. The received modbus commands work on memory buffer whose address is defined in the **Buffer** and its size in bytes is defined in **Size**.



In **IFTime** must be defined the interframe time of modbus command. This is the time that elapses between the receiving frame and the next frame. On serial line this time coincides with the time of receiving 3 characters at defined baud rate.

On every received command, the **Done** output is activated for a program loop. If there is a command error, the **Fault** output will be activated for a program loop and the **Errors** value will be increased.

Enable (BOOL)	Enable function block.
File (FILEP)	Stream returned from Sysopen function.
Type (USINT)	Modbus type selector. 0:RTU, 1:Ascii, 2:TCP
Node (USINT)	Modbus node number (Range from 0 to 255).
Offset (UINT)	Offset on received modbus frame address (Range from 16#0000 to 16#FFFF).
Buffer (@USINT)	Address of buffer where the modbus commands read or write.
Size (UINT)	Size in bytes of data buffer where the modbus commands read or write.
IFTime (UDINT)	Time between frame (μ S). If you use the serial port, this time must be related to baud rate.

Baud rate	Time
300	112000
600	56000
1200	28000
2400	14000
4800	7000
9600	3430

Baud rate	Time
19200	1720
38400	860
57600	573
76800	429
115200	286

On ethernet connection set the minimum time value.

Done (BOOL)	Active for a program loop when a modbus command is received.
Fault (BOOL)	Active for a program loop if there is an error receiving modbus command.
Errors (UDINT)	Increased at every error. If it reach the maximum value, it restarts from 0.

Supported command

The function block only supports a few commands provided by the Modbus protocol. The commands supported are:

Code	Description
01	Read coil status (Max 250 coils)
02	Read input status (Max 250 coils)
03	Read holding registers (Max 125 registers)
04	Read input registers (Max 125 registers)
05	Force single coil
06	Preset single register
08	Loopback diagnostic test
0F	Force multiple coils (Max 250 coils)
10	Preset multiple registers (Max 125 registers)

Error codes

If an error occurs, the **Fault** output is activated and [SysGetLastError](#) can detect the error code. In case of modbus command exception, the error code was reported without activate the **Fault**.

- 10038010 **File** value not defined.
- 10038050 Execution timeout.
- 10038060 Execution error.
- 10038080 **Type** definition error.
- 10038100~19 Received frame error (Frame length error).
- 10038150~1 Received frame error (Wrong CRC).
- 10038200~2 Error transmitting the answer frame.
- 10038501 Exception 01. **Illegal function**, received command non supported.
- 10038502 Exception 02. **Illegal data address**, received command with address or number of data out of range.
- 10038503 Exception 03. **Illegal data value**, received command with data out of range.
- 10038504 Exception 04. **Failure in associated device**, received command with errors.

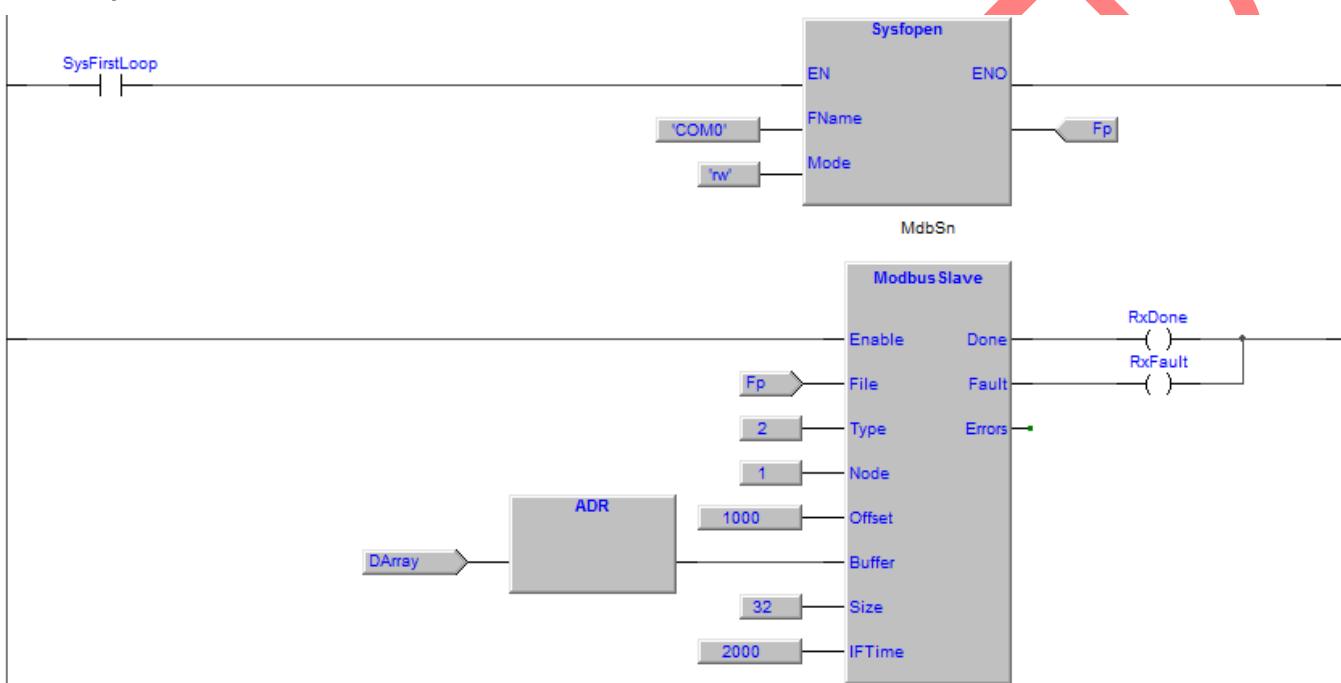
Examples

Modbus RTU slave protocol is run on **COM0**. It uses the default serial settings **115200,e,8,1**. The Modbus commands can act on the **DArray** WORD array.

Defining variables

	Name	Type	Address	Array	Initvalue	Attribute	Description
1	RxDone	BOOL	Auto	No	FALSE	..	Modbus Rx command Ok
2	RxFault	BOOL	Auto	No	FALSE	..	Modbus Rx command fault
3	Fp	FILEP	Auto	No	0	..	File pointer
4	MdbSn	ModbusSlave	Auto	No	0	..	Modbus RTU slave FB
5	DArray	WORD	Auto	[0..31]	32(0)	..	Modbus accessible area

LD example (PTP114A610, LD_ModbusSlave)



7.13.5 OnOffCycle_v1, on/off cycle with random times

Type	Library
FB	eLLabUtyLib_C030

This function block performs the timing of an on/off cycle with random On and Off time defined between a minimum and maximum value. The maximum managed value is up to 1193 hours.

Activating the **Enable** command, the **Out** output performs a blink on/off with a random time between the minimum and maximum values defined. Disabling the input, the Out output is deactivated.

The **Delay** variable return the actual delay time. The **Time** variable returns the time countdown.



Enable (BOOL) Enable function block.

MinOffTime (UDINT) Minimum value for command off time (mS).

MaxOffTime (UDINT) Maximum value for command off time (mS).

MinOnTime (UDINT) Minimum value for command on time (mS).

MaxOnTime (UDINT) Maximum value for command on time (mS).

Out (BOOL) On/Off command status.

Delay (UDINT) Actual delay time used for temporization (mS).

Time (UDINT) Temporization countdown time (mS).

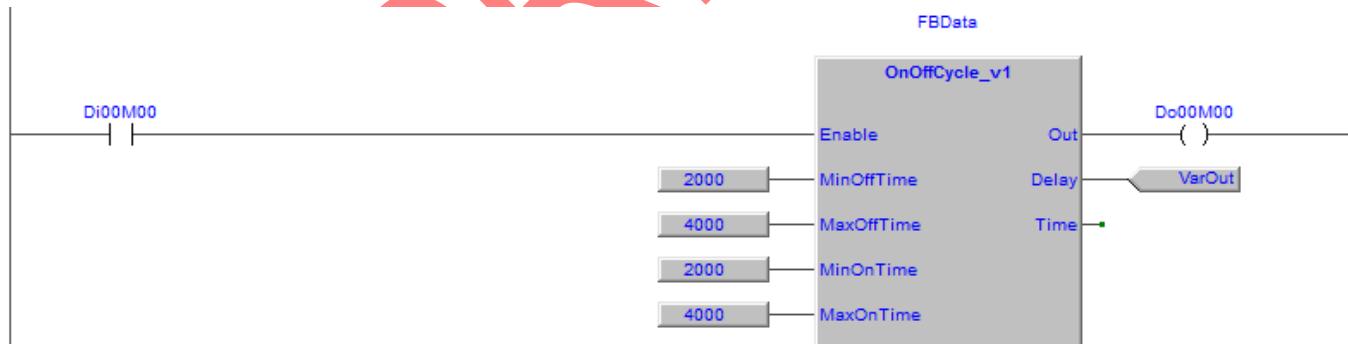
Example

It is executed the blinking of the **Do00M00** output with random times between 2 and 4 seconds.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	FBDData	OnOffCycle_v1	Auto	No	On/Off cycle FB data
2	VarOut	UDINT	Auto	No	Variable output

LD example (PTP114A640, LD_OnOffCycle_v1)



IL example (PTP114A640, IL_OnOffCycle_v1)

```

CAL FBDData (* Call the ONOFFCYCLE function block *)

LD Di00M00
ST FBDData.Enable (* Transfer the digital input to Enable input *)

LD 2000
ST FBDData.MinOffTime (* Set the minimum off time *)

LD 4000
ST FBDData.MaxOffTime (* Set the maximum off time *)

LD 2000
ST FBDData.MinOnTime (* Set the minimum on time *)

```

```
LD  4000
ST  FBData.MaxOnTime (* Set the maximum on time *)

LD  FBData.Out
ST  Do00M00 (* Copy the Out value to logic output *)

LD  FBData.Delay
ST  VarOut (* The Delay time is copied to variable *)
```

ST example (PTP114A640, ST_OnOffCycle_v1)

```
FBData(); (* Call the ONOFFCYCLE function block *)

FBData.Enable:=Di00M00; (* Transfer the digital input to Enable input *)
FBData.MinOffTime:=2000; (* Set the minimum off time *)
FBData.MaxOffTime:=4000; (* Set the maximum off time *)
FBData.MinOnTime:=2000; (* Set the minimum on time *)
FBData.MaxOnTime:=4000; (* Set the maximum on time *)

Do00M00:=FBData.Out; (* Copy the Out value to logic output *)
VarOut:=FBData.Delay; (* The Delay time is copied to variable *)
```

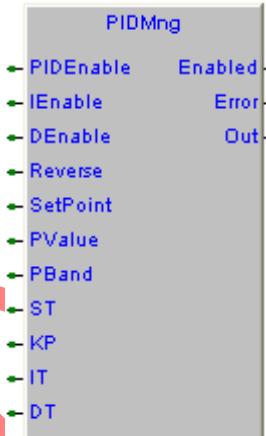
DEPRECATED

7.13.6 PIDMng, PID management

Type	Library
FB	eLLabUtyLib_C030

This function block executes the PID regulation. It is possible to enable individually the action to use: (**P**roportional (**I**ntegrative, (**D**erivative.

The **Reverse** command, allows to invert the sign of the **Out** output.



PIDEnable (BOOL)	Enable PID control. By activating the input, the regulation is enabled. Disabling input resets the Out output value.
IEnable (BOOL)	Enable integrative regulation. By activating the input, the integrative regulation is enabled.
DEnable (BOOL)	Enable derivative regulation. By activating the input, the derivative regulation is enabled.
Reverse (BOOL)	By activating the input, the Out output value is inverted.
SetPoint (REAL)	Set point. The value is expressed in the unity of measure of process to control.
PValue (REAL)	Process variable. The value is expressed in the unity of measure of process to control.
PBand (REAL)	Proportional band. This value defines the value of error beyond which the control is disabled and the Out output is forced to ±100% . The value is expressed in the unity of measure of process to control.
ST (REAL)	Scansion time. You need to set the time used to execute the integrative and derivative regulation if enabled. The value is in mS .
KP (REAL)	Proportional constant. Please note that with higher values, there is a more ready regulation with a consequent increase of the overshoot value. The value is a number.
IT (REAL)	Integrative time. Please note that with higher values it is less fast to regain error. The value is in Sec .
DT (REAL)	Derivative time. Please note that with higher values it is more fast to regain error. The value is in Sec .
Enabled (BOOL)	PID regulation enabled.
Error (BOOL)	Execution error.
OUT (REAL)	Correction value in output from PID. This value must be used to control the process. The value is expressed in %. The range is between 0 and ±100% .

Error codes

If an error occurs, the **Error** output is activated and [**SysGetLastError**](#) can detect the error code.

10012050 Not defined value for **ST**.

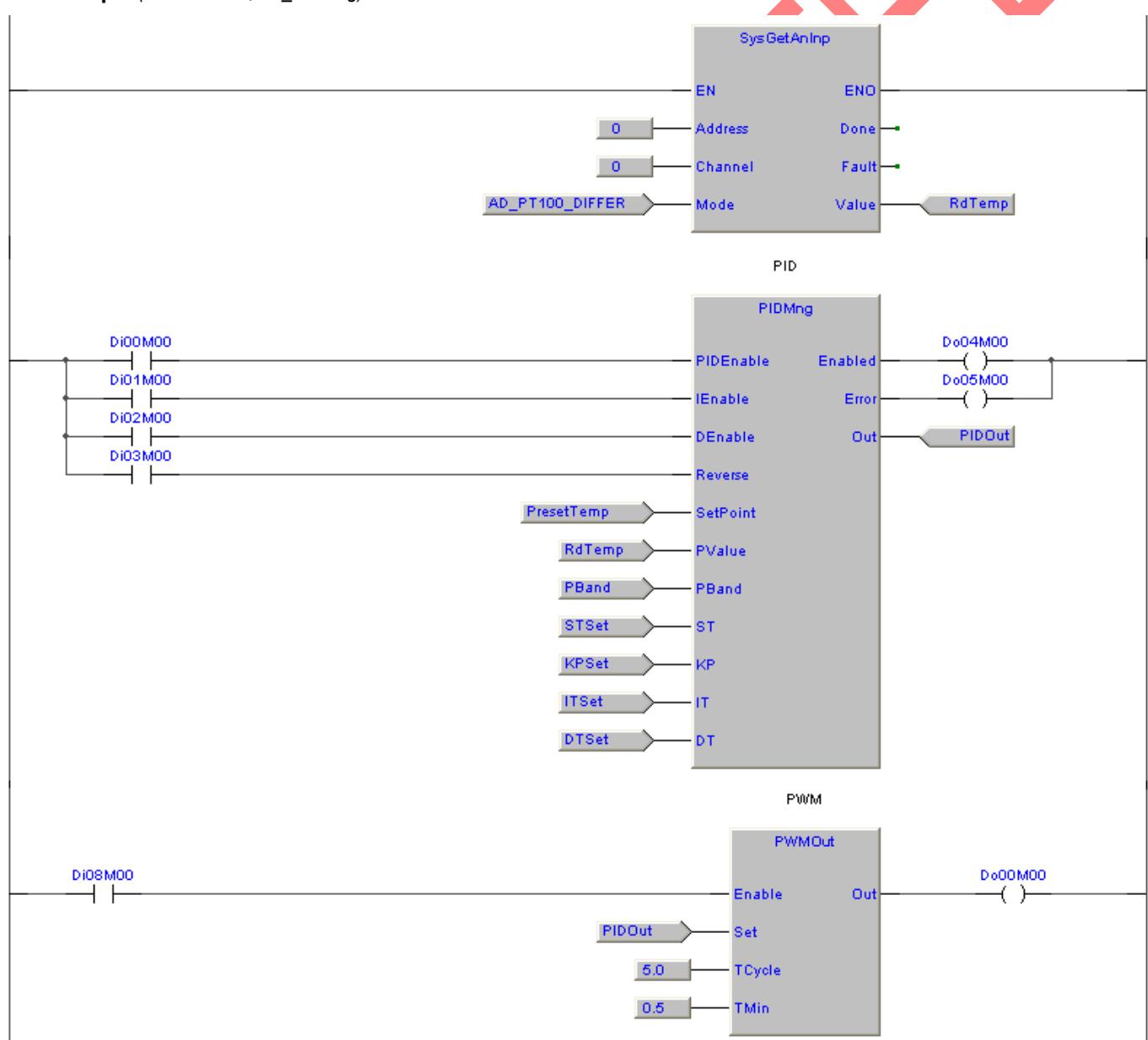
Examples

In the example a temperature control is managed on a heater. A Pt100 temperature sensor is acquired and a PWM output through **Do00M00**, is managed. The constants of the PID loop are allocated in backup so they are maintained at power off. They are also accessible via Modbus.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	RdTemp	REAL	%MD100.0	No	0	..	Read temperature (Degrees)
2	PresetTemp	REAL	%MD100.2048	No	0	..	Set point temperature (Degrees)
3	PBand	REAL	%MD100.2052	No	0	..	Proportional band (Degrees)
4	STSet	REAL	%MD100.2056	No	0	..	Scansion time (mS)
5	KPSet	REAL	%MD100.2060	No	0	..	Proportional coefficient
6	ITSet	REAL	%MD100.2064	No	0	..	Integrative time (S)
7	DTSet	REAL	%MD100.2068	No	0	..	Derivative time (S)
8	TempRead	SysGetAnInp	Auto	No	0	..	Analog input
9	PWM	PWMOut	Auto	No	0	..	PWM management
10	PIDOut	REAL	Auto	No	0	..	PID output value (%)
11	PID	PIDMng	Auto	No	0	..	PID management

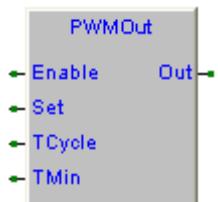
LD example (PTP114A100, LD_PIDMng)



7.13.7 PWMOut, PWM output management

Type	Library
FB	eLLabUtyLib_C030

This function block performs the management of a PWM output.



- Enable** (BOOL) Enable PWM output management. Activating the input, the management is enabled. Deactivating the input, the **Out** output is reset.
- SET** (REAL) PWM set value. It is expressed in %.
- TCycle** (REAL) PWM cycle time. Value is expressed in S.
- TMin** (REAL) Minimum **Out** command time. Value is expressed in S.
- Out** (BOOL) PWM output.

Examples

In the example is managed a PWM output defining a cycle time of 5 seconds with a minimum time of 0.5 seconds. Setting the set point value and 50% and activating the **Di00M00** input, the **Do00M00** output will be activated for 2.5 seconds and deactivated for 2.5 seconds.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	FBData	PWMOut	Auto	No	0	..	FB PWMOut data

LD example (PTP114A100, LD_PWMOut)



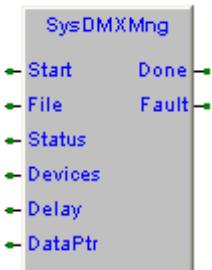
7.13.8 SysDMXMng, DMX management

Type	Library
FB	XTarget_07_0

This function block performs the management of the DMX protocol. This function block is protected and require a code to unlock it (see [functions and function blocks protection](#)). It is possible to use it freely in test mode for 15 min.

Activating the **Start** input, over the serial file identified by **File**, is sent a DMX frame that begins with the value of **Status** and follows with the preset value of the devices defined by **Devices**. The preset value of the various devices must be loaded into an array of data whose address is passed in **DataPtr**. Keeping active the **Start** input, DMX frames are sent consecutively.

After sending the DMX command, the **Done** output will be activated for a program loop.



Start (BOOL)	Send DMX command frame to serial port. It resets automatically when the frame is sent.
File (FILEP)	Stream returned by Sysfopen function.
Status (USINT)	Value of status byte sent as first byte of DMX frame, before the preset devices bytes.
Devices (UINT)	Number of devices connected to DMX bus.
Delay (UINT)	Wait time between DMX frames transmission (mSec)
DataPtr (@USINT)	Pointer to array of DMS devices preset values.
Done (BOOL)	Active for a program loop when a DMX frame is sent.
Fault (BOOL)	Active if there is a management error.

Error codes

If an error occurs, the **Fault** output is activated and [SysGetLastError](#) can detect the error code.

- 9979050 Function block allocation error.
- 9979060 Relocatable memory space full. You can not run the FB.
- 9979070 Function block version error.
- 9979085 Protected FB. The available time for demo mode is over.
- 9979200 DMX protocol not supported by device defined with **File**.
- 9979990 Not yet Implemented in the simulator.

Examples

Having to handle pointers to memory, it is preferable to use the function block within a ST program. The example enables the DMX protocol on the serial port **COM1**. Five devices are managed with addresses from 1 to 5. The DMX frame is continuously sent to the devices.

Activating the **Di00M00** digital input, the value is set to 0 on all devices. Activating the **Di01M00** digital input on the device 1 is set a value 10, on the 2 the value 20, and so on until the fifth where the value is 50.

Defining variables

	Name	Type	Address	Array	Initvalue	Attribute	Description
1	FBDMX	SysDMXMng	Auto	No	0	..	FB gestione protocollo DMX
2	DMXData	USINT	Auto	[0..4]	5(0)	..	DMX data
3	DiPls	R_TRIG	Auto	[0..1]	0,0	..	Pulse su ingresso

ST example

```

(* ----- *)
(* SERIAL PORT OPEN *)
(* ----- *)
(* Here the COM1 port is opened in read/write. *)

IF (FBDMX.File = NULL) THEN
    FBDMX.File:=Sysopen('COM1', 'rw'); (* Port COM1 file pointer *)
END_IF;

(* ----- *)
(* COMMANDS ACTIVATION *)
(* ----- *)
(* Commands activation on Di00M00 input. *)

DiPls[0](CLK:=Di00M00);
IF (DiPls[0].Q) THEN
    DMXData[0]:=0; (* Preset device with address 1 *)
    DMXData[1]:=0; (* Preset device with address 2 *)
    DMXData[2]:=0; (* Preset device with address 3 *)
    DMXData[3]:=0; (* Preset device with address 4 *)
    DMXData[4]:=0; (* Preset device with address 5 *)
    FBDMX.Start:=TRUE; (* Start *)
END_IF;

(* Commands activation on Di01M00 input. *)

DiPls[1](CLK:=Di01M00);
IF (DiPls[1].Q) THEN
    DMXData[0]:=10; (* Preset device with address 1 *)
    DMXData[1]:=20; (* Preset device with address 2 *)
    DMXData[2]:=30; (* Preset device with address 3 *)
    DMXData[3]:=40; (* Preset device with address 4 *)
    DMXData[4]:=50; (* Preset device with address 5 *)
    FBDMX.Start:=TRUE; (* Start *)
END_IF;

(* ----- *)
(* DMX PROTOCOL MANAGEMENT *)
(* ----- *)
(* DMX protocol management. *)

FBDMX.Status:=0; (* Status byte *)
FBDMX.Devices:=5; (* Number of devices *)
FBDMX.Delay:=0; (* Interframe delay (mSec) *)
FBDMX.DataPtr:=ADR(DMXData); (* Data array pointer *)
FBDMX(); (* FB DMX *)
IF (FBDMX.Done) THEN FBDMX.Start:=FALSE; END_IF;

(* [End of file] *)

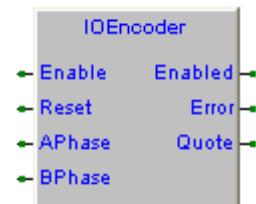
```

7.13.9 IOEncoder, incremental encoder over I/O

Type	Library
FB	eLLabUtyLib_C030

This function block reads of an incremental encoder connected to the logic inputs. Simply place the two digital input used as channel **A** and **B** of incremental encoder, to the two **APhase** and **BPhase** input. The function block performs the quadrature signals, control of rotation direction and manages the **Quote** value in output.

The quadrature signals performs the multiplication by 4 of encoder notches, so the **Quote** value at the end of a complete revolution of the encoder is equal to the number of encoder notches multiplied by 4.



Enable (BOOL) Enable the encoder counter management.

Reset (BOOL) Reset command. Activating it, the **Quote** value is reset.

APhase (BOOL) Encoder channel **A**.

BPhase (BOOL) Encoder channel **B**.

Enabled (BOOL) Encoder counter enabled.

Error (BOOL) Active for a program loop when there is an encoder acquisition error. It is activated if the frequency of the encoder signals are bigger than the execution time of the function block.

Quote (UDINT) Encoder quote expressed in impulses. It is the number of notches/revolution multiplied by 4.

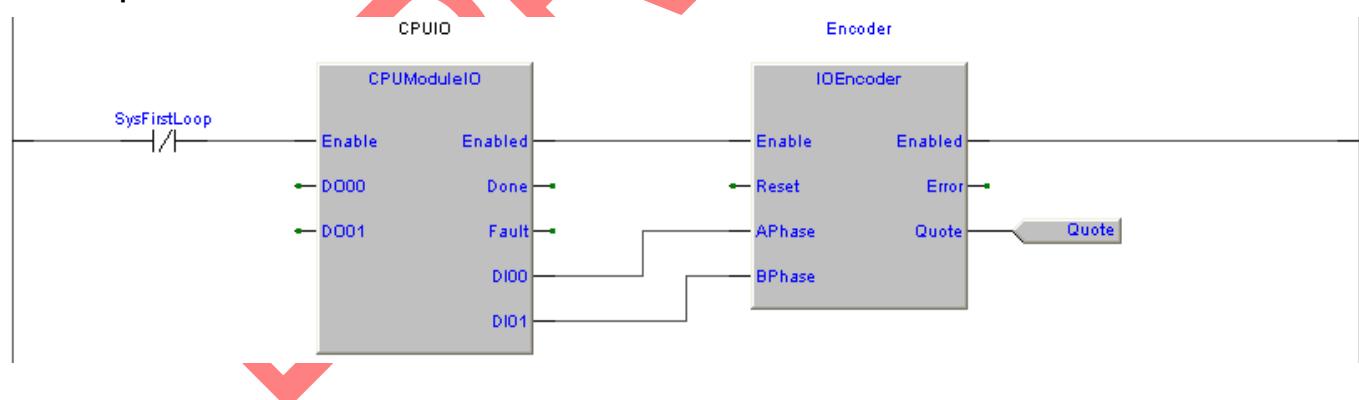
Examples

The example manages the acquisition of an incremental encoder connected to the inputs of the CPU module. Turning the encoder for one revolution, the **Quote** value will be incremented if the rotation is clockwise (CW) or decremented if the rotation is counterclockwise (CCW), of the number of notches per revolution multiplied by 4.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Encoder	IOEncoder	Auto	No	0	..	IOEncoder function block
2	CPUIO	CPUModuleIO	Auto	No	0	..	CPUModuleIO function block
3	Quote	UDINT	Auto	No	0	..	Encoder quote

LD example



7.13.10 GetISO1155Crc, calculate CRC according ISO1155

Type	Library
Function	eLLabUtyLib_C030

This function calculates the CRC **Cyclic Redundancy Check** on a data area. This is calculated in accordance with **ISO 1155**.

You must pass the address of the memory buffer **Buffer** and the number of bytes **ByteNr** on which to perform the CRC calculation.



Buffer (@USINT) Address of memory buffer on which to execute the CRC calculation.

ByteNr (UINT) Number of bytes on which to execute the CRC calculation, starting from the address defined in **Buffer**.

Result (UINT) Calculated CRC value.

Example

It is calculated the CRC of a read request from the registry 1.8.1 from an electrical energy meter according to IEC 62056-2. The request frame is '**<SOH>R1<STX>1.8.1()<ETX><CRC>**'.

The CRC value returned in **CRCValue** is 16#5A (90 decimal).

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	DataFrame	STRING	Auto	[16]	..		Data frame
2	CRCValue	UINT	Auto	No	0	..	CRC Value

ST example

```

(* **** IEC1155 CRC CALCULATION **** *)
(* Register read command '<SOH>R1<STX>1.8.1()<ETX><CRC>'. *)
Dataframe:='\$01R1\$021.8.1()\$03'; (* Data frame *)
CRCValue:=GetISO1155Crc(ADR(Dataframe), 12); (* CRC Value *)
  
```

7.13.11 IODataExchange, exchange data by using logic I/O

Type	Library
FB	eLLabUtyLib_C030

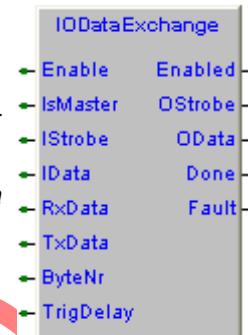
This function block allows data exchange between two systems, one master and one slave, using a connection with I/O logic. They used two inputs and two outputs for each system. You can define the number of bytes of data to be exchanged.

You must connect the **OStrobe** digital output of one system to the **IStrobe** digital input of other system and the **OData** to the **IData** of the other.

Data transfer is bidirectional: data in the **TxData** buffer of a system are transferred into the **RxData** buffer of the other system and vice versa, for the number of bytes defined by **ByteNr**. Communication is verified by sending a CRC according to ISO 1155 standard.

At the end of each data transfer, the **Done** output is activated for a program loop. On its activation, you must transfer the data to be transmitted in the transmission buffer, and read data from the receive buffer.

In case of **error** in communication, the **Fault** output will be activated for a program loop, and the two systems resynchronize to resume a new transmission.



Enable (BOOL) Communication enable.

IsMaster (BOOL) **TRUE**: Master mode, **FALSE**: Slave mode.

IStrobe (BOOL) Set to strobe digital input.

IData (BOOL) Set to data digital input.

RxData (UDINT) Buffer address of received data.

TxData (UDINT) Buffer address of data to send.

ByteNr (USINT) Number of bytes to exchange with other system (From 1 to 30).

TrigDelay (UINT) Time between out of **OData** and out of **OStrobe** (From 0 to 30 mS).

OStrobe (BOOL) Set to strobe digital output.

OData (BOOL) Set to data digital output.

Done (BOOL) Active for a program loop at the end of data exchange.

Fault (BOOL) Active for a program loop if data exchange error.

Error codes

If an error occurs, the **Fault** output is activated and [SysGetLastError](#) can detect the error code.

10011080 Error defining **ByteNr** value.

10011082 Error defining **TrigDelay** value.

10011100~1 Timeout on waiting **IStrobe** signal activation.

10011110~1 Timeout on waiting **IStrobe** signal deactivation.

10011200~1 Wrong CRC of received data.

Examples

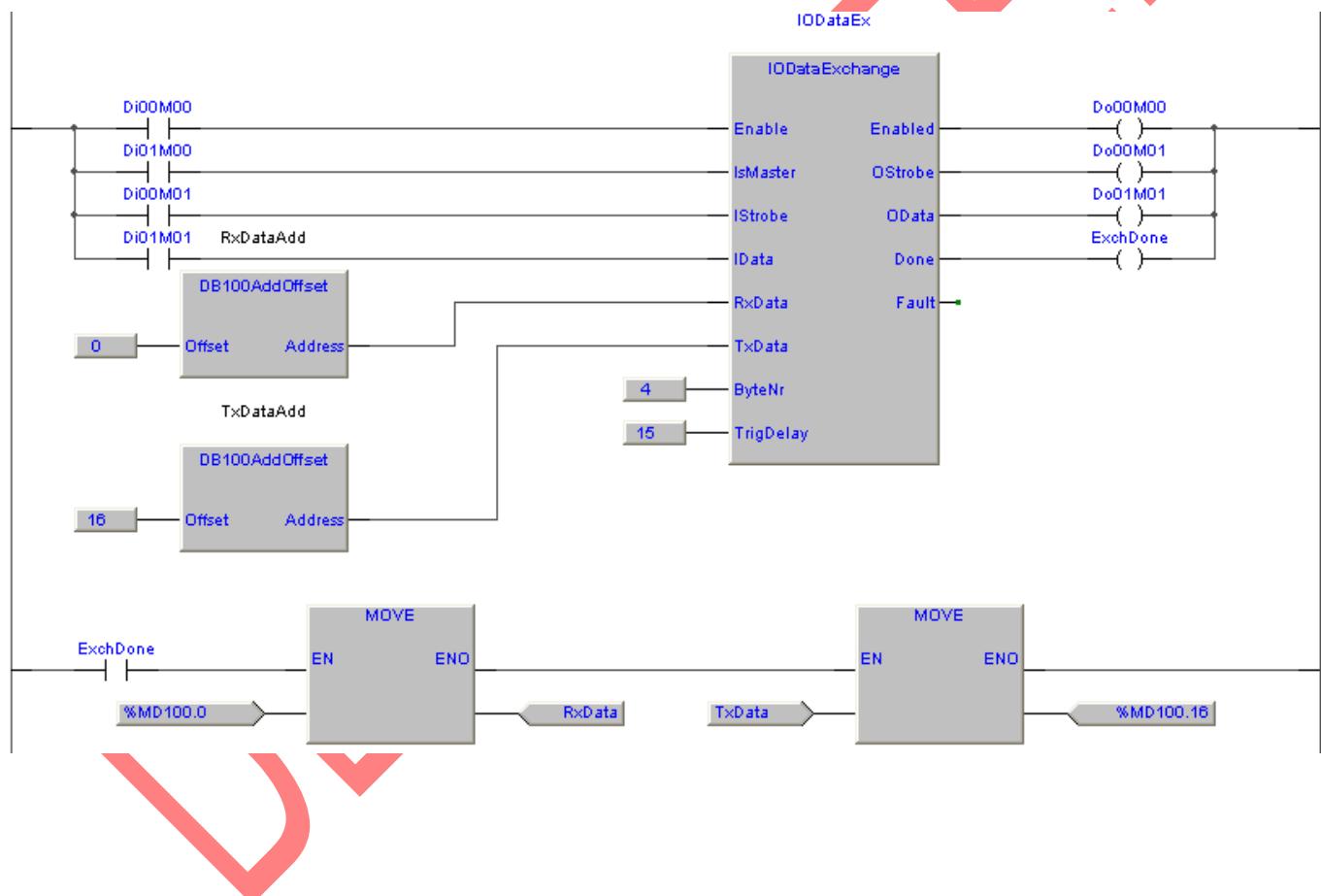
Using two systems and activating the master mode (**Di01M00** active) on the first and the slave mode (**Di01M00** off) on the other, it is possible to exchange 4 bytes of memory between systems. The 4 bytes are allocated at address **MD100.0** of the first system, will be transferred to 4 bytes allocated to address **MD100.16** of the other system.

After the transfer, the received data are transferred from memory **MD100.0** to the **RxData** variable, while the **TxData** variable is transferred in memory **MD100.16**.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	ExchDone	BOOL	Auto	No	FALSE	..	Data exchange done
2	RxData	UDINT	Auto	No	0	..	Received data
3	TxDATA	UDINT	Auto	No	0	..	Transmit data
4	RxDATAAdd	DB100AddOffset	Auto	No	0	..	RxDATA address
5	TxDATAAdd	DB100AddOffset	Auto	No	0	..	TxDATA address
6	IODataEx	IODataExchange	Auto	No	0	..	FB IODataExchange data

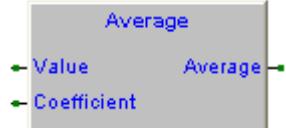
LD example (Ptp121A000)



7.13.12 Average, value average

Type	Library
FB	eLLabUtyLib_C030

This function block performs the average on a value. The average action is defined by a parameter **Coefficient**. An higher value for the parameter, increases the average action on the **Average** output value.



Value (REAL) Value on which to do the average

Coefficient (REAL) Value of the average coefficient.

Average (REAL) Average output value.

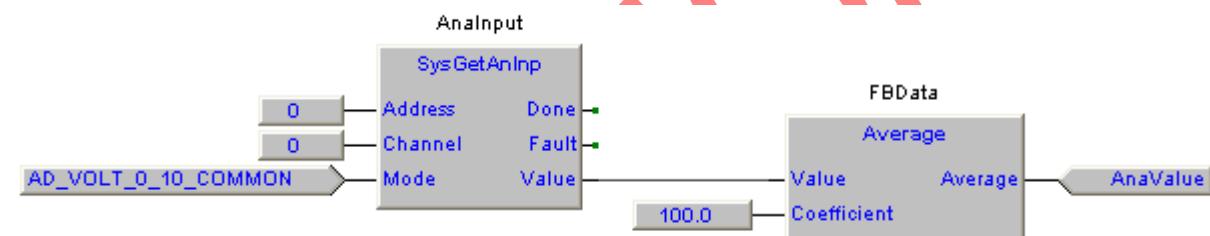
Examples

It is executed an analog acquisition from the channel **0** of the module with address **0**, in 0÷10 volts mode. The acquired value is averaged and then transferred into the **AnaValue** variable.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	FBData	Average	Auto	No	0	..	FB average data
2	AnalInput	SysGetAnInp	Auto	No	0	..	FB Analog input data
3	AnaValue	REAL	Auto	No	0	..	Analog input value

FBD example (PTP114A500, FBD_Average)



7.13.13 HIDClkDtaReader, HID RFID clock/data reader

Type	Library
FB	eLLabUtyLib_C030

This function block executes the acquisition of an HID reader clock and data type. The FB must be executed at least every 400 uSec, so we recommend to run the program the Fast task, defining the execution time of the task through to 400 uS [SysSetTaskLpTime](#) function.

The function block can acquire RFID tags with code length up to 8 bytes, the code read is transferred into the buffer addressed by **CData**, you must define the length of the code in **CDLength**.



Clock (BOOL) RFID reader clock signal.

Data (BOOL) RFID reader data signal.

CData (@BYTE) RFID tag code buffer.

CDLength (USINT) RFID tag code length.

Done (BOOL) Set for a program loop when RFID code is read.

Fault (BOOL) Set for a program on RFID code read error.

Error codes

If an error occurs, the **Fault** output is activated and [SysGetLastError](#) can detect the error code.

10039050 **CDLength** value error.

10039060 Execution error.

10039100~2 RFID tag read error.

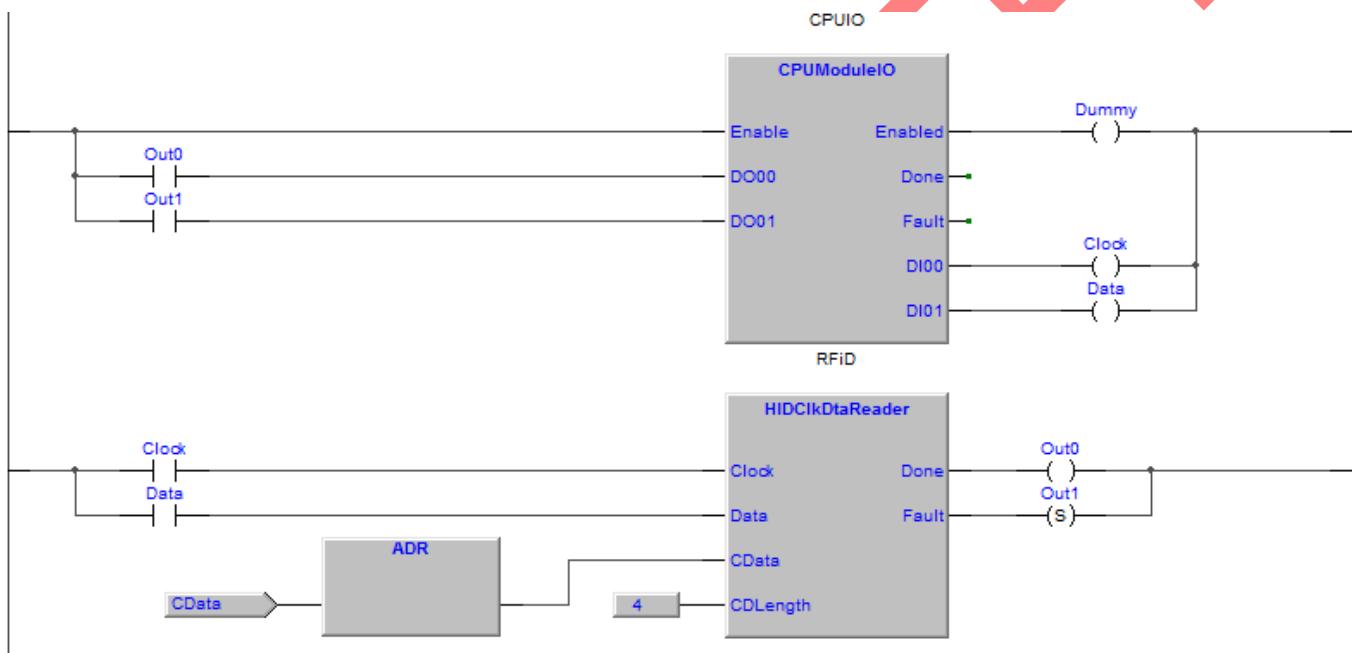
Examples

It's managed to read an HID RFID clock and data reader connected to the CPU module. I remember that the program must be executed at least every 400 uS, so it must be executed in the Fast task, setting its execution time to 400 uS. The code value read by the RFID tag is transferred **CData** array.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Clock	BOOL	Auto	No	FALSE	..	Clock input signal
2	Data	BOOL	Auto	No	FALSE	..	Data input signal
3	Dummy	BOOL	Auto	No	FALSE	..	Dummy variable
4	Out0	BOOL	Auto	No	FALSE	..	Output 0 on CPU module
5	Out1	BOOL	Auto	No	FALSE	..	Output 1 on CPU module
6	CData	BYTE	Auto	[0..7]	8(0)	..	Card data
7	CPUIO	CPUModuleIO	Auto	No	0	..	CPUModuleIO function block
8	RFID	HIDClikDtaReader	Auto	No	0	..	RFID read

LD example (PTP114A610, LD_HIDCardRead)



7.13.14 Linearize, linearize a non linear value

Type	Library
Function	eLLabUtyLib_C030

This function performs the linearization of a value. Must be provided to the function the address definition array of the input value to be linearized **VInReference**, the address definition array of the corresponding output value linearized **VOutReference** and the number of values in the arrays **ReferenceNr**. It's important to define the data in the two arrays in ascending order, starting from the smallest value.

The function searches in the **VInReference** array a value immediately higher than **VIn** and interpolates between the value found and the previous one, calculating the output value according to the two values in the same locations in the **VOutReference**.



- VIn** (REAL) Value to be linearized.
VInReference (@REAL) Pointer to the array that defines the input value to be linearized.
VOutReference (@REAL) Pointer to the array that defines the output value linearized.
ReferenceNr (USINT)
(REAL) Number of values in the linearization arrays.
Linearized output value

Examples

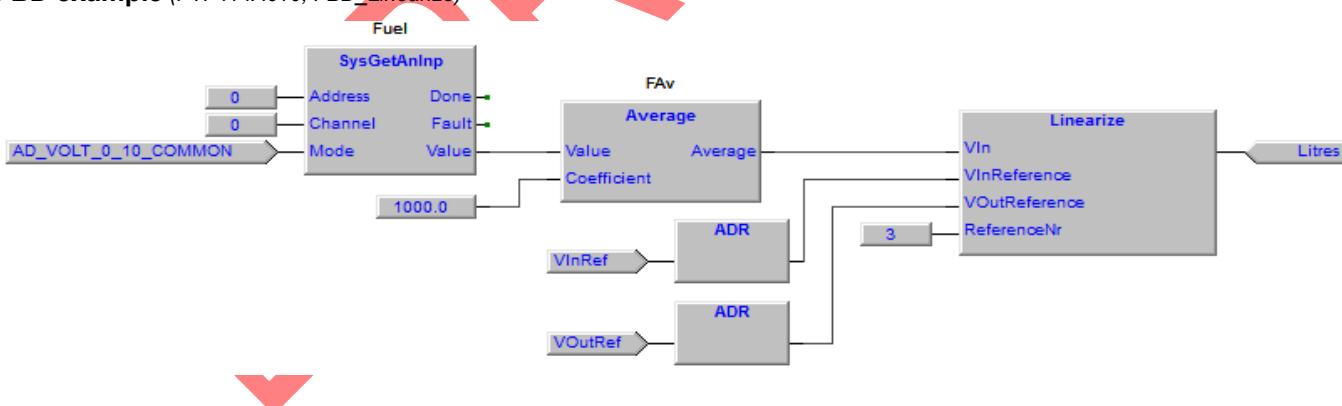
It's performed an analog reading of a tank level, the tank has an irregular shape, through the function it's returned the value of liters in the tank. The level value is averaged to have a more regular value.

1 volt corresponds to 1000 liters, 2 volts correspond to 4000 liters, 3 volts are 6000 liters.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Fuel	SysGetAnInp	Auto	No	0	..	Analog acquisition
2	Litres	REAL	Auto	No	0	..	Fuel quantity (Litres)
3	VInRef	REAL	Auto	[0..15]	1,2,3,13(0)	..	Level voltage input
4	VOutRef	REAL	Auto	[0..15]	1000,4000,6000,13(0)	..	Litres value
5	FAv	Average	Auto	No	0	..	Level average

FBD example (PTP114A610, FBD_Linearize)

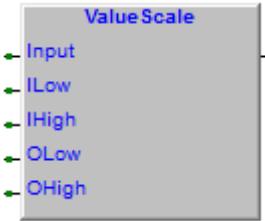


7.13.15 ValueScale, scales a value

Type	Library
Function	eLLabUtyLib_C030

This function executes a value scaling. It's possible to define the minimum and maximum input value to be scaled (**ILow**, **IHigh**) and the minimum and maximum output scaled value (**OLow**, **OHigh**).

The use of this function is particularly suitable to converting values read from (4-20 mA) current sensors in the corresponding measurement units detected by the sensor.



Input (REAL) Input value to be scaled.

ILow (REAL) Low limit of the input value to be scaled.

IHigh (REAL) High limit of the input value to be scaled.

OLow (REAL) Low limit of the scaled output value.

OHigh (REAL) High limit of the scaled output value.

(REAL) Scaled output value.

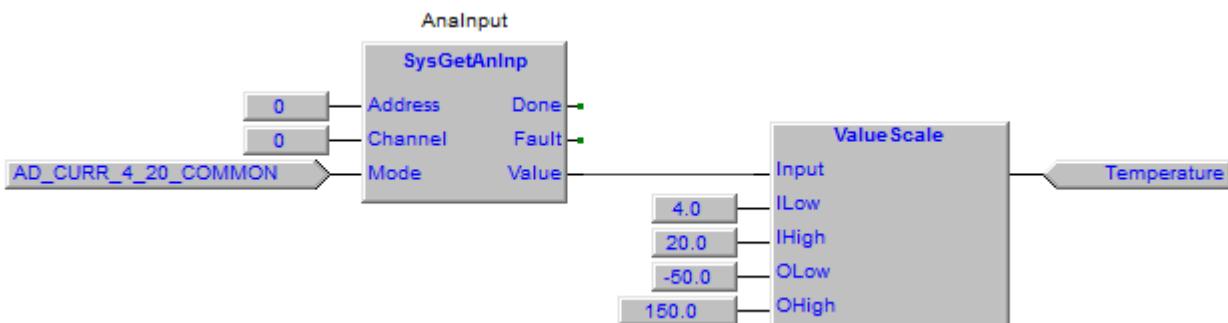
Examples

It's executed an analog acquisition of a temperature sensor with a 4-20 mA current output, the sensor has the output value of 4 mA at -50 °C and 20 mA to 150 °C. Using the function the output current is directly converted in the temperature value.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	AnalInput	SysGetAnInp	Auto	No	0	..	FB Analog input data
2	Temperature	REAL	Auto	No	0	..	Temperature value (°C)

FBD example (PTP114A610, FBD_Scale)

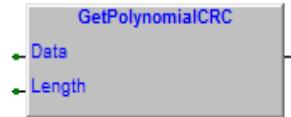


7.13.16 GetPolynomialCRC, polynomial CRC calculation

Type	Library
Function	eLLabUtyLib_C030

This function calculates the CRC **Cyclic Redundancy Check** on a data area. This is calculated in accordance with **CCITT**.

You must pass the address of the memory buffer **Buffer** and the number of bytes **Length** on which to perform the CRC calculation.



- Data** (@USINT) Address of memory buffer on which to execute the CRC calculation.
Length (UINT) Number of bytes on which to execute the CRC calculation, starting from the address defined in **Data**.
Result (WORD) Calculated CRC value.

Examples

The CRC calculation is performed on an array of data. Suppose you have an array of 8 bytes allocated in DB100 at address 16 which contains the values 16#04, 16#3B, 16#1B, 16#00, 16#00, 16#00, 16#00, 16#00 , the calculated CRC will be 16#5AF0.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	CRCValue	WORD	Auto	No	CRC calculated value
2	BData	BYTE	Auto	[0..7]	Byte data

ST example (PTP114A620, ST_GetPolynomial/CRC)

```

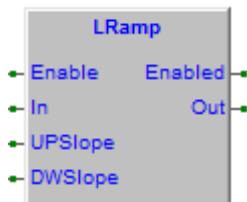
BData[0]:=16#04;
BData[1]:=16#3B;
BData[2]:=16#1B;
BData[3]:=16#00;
BData[4]:=16#00;
BData[5]:=16#00;
BData[6]:=16#00;
BData[7]:=16#00;
CRCValue:=GetPolynomialCRC(ADR(BData), 8);
  
```

7.13.17 LRamp, linear ramp

Type	Library
FB	eLLabUtyLib_C030

This function block executes a linear ramp on the input value. The output value **Out** follows the input value **In** with slope values defined. The **UPSlope** value defines the ramp slope in raising, the **DWSlope** value defines the ramp slope in falling. Disabling the function block the output is set to zero.

The slope values are defined in units per second, for example with **UPSlope** set to 1 (1 unit per second), assuming a step of 10 input **In**, we will have the output **Out** that reaches the value 10 in 10 seconds.



Enable (BOOL) FB enable.

In (REAL) Input value on which it is referred the output ramp.

UPSlope (REAL) Up slope ramp coefficient (Unit/Second)

DWSlope (REAL) Down slope ramp coefficient (Unit/Second)

Enable (BOOL) FB enabled.

Out (REAL) Output ramp value.

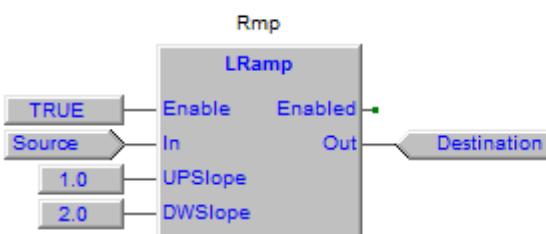
Examples

Is performed on a ramp value of Source with up slope of 1 and down slope of 2. Assuming a step of 10 in Source the Destination will reaches the value on 10 seconds. Setting **Source** to 0 the **Destination** will reach the 0 on 5 seconds.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Source	REAL	Auto	No	Source value
2	Destination	REAL	Auto	No	Destination value
3	Rmp	LRamp	Auto	No	FB instance

FBD example



7.13.18 VaPotentiometer, voltage acquisition potentiometer

Type	Library
Function	eLLabUtyLib_C030

This function calculates the ohmic value of a potentiometer acquiring the value of output voltage from its cursor.

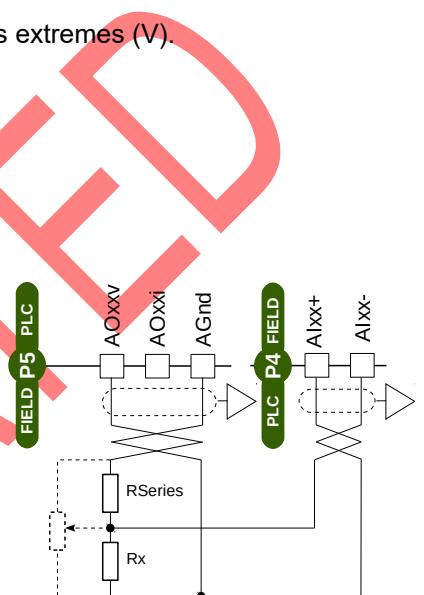
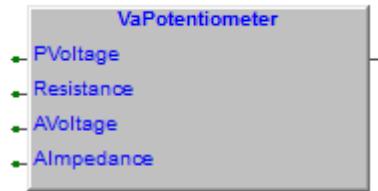
In **PVoltage** must be indicate the potentiometer supply voltage, **Resistance** is the potentiometer resistance. Providing in **AVoltage** the voltage value readed from its slide the function calculates the resistance value of the potentiometer to its current location.

To compensate the reading error due to the input impedance of the voltage acquisition channel necessary to define it in **Almpedance**.

- PVoltage** (REAL) Supply voltage of the potentiometer. The voltage applied to its extremes (V).
- Resistance** (REAL) Resistance value of the potentiometer (Ohm).
- AVoltage** (REAL) Voltage value acquired on the potentiometer slider (V).
- Almpedance** (REAL) Value of input impedance of the acquisition channel (Ohm).
- (REAL) Value of potentiometer resistance (Ohm).

Examples

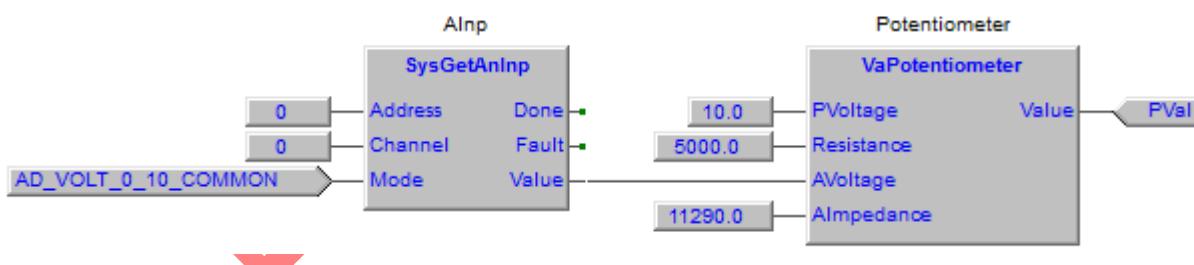
It's acquired the ohmic value at the cursor position of a 5 kOhm potentiometer powered with a voltage of 10 volts. The value is acquired through an analog input module.



Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Alnp	SysGetAnInp	Auto	No	..		Analog input FB
2	Potentiometer	VaPotentiometer	Auto	No	..		Potentiometer acquisition FB
3	PVal	REAL	Auto	No	..		Potentiometer value (Ohm)

FBD example (PTP114A640, FBD_VaPotentiometer)



7.13.19 ResistorValue, resistor value acquire

Type	Library
Function	eLLabUtyLib_C030

This function calculates the ohmic value of a resistor acquiring the value of output voltage from a resistor divider.

In **PVoltage** must be indicate the resistor divider supply voltage, **RSeries** is the Ohm value of the series resistor. Providing in **AVoltage** the voltage value readed on the resistor to be misured, the function calculates the resistance value.

To compensate the reading error due to the input impedance of the voltage acquisition channel necessary to define it in **Almpedance**.



PVoltage (REAL) Supply voltage of the resistor divider (V)

RSeries (REAL) Series resistor value (Ohm).

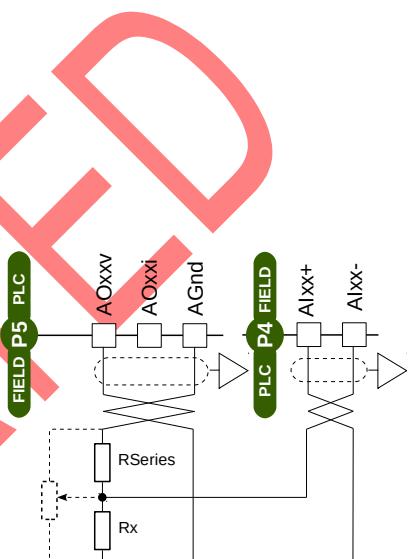
AVoltage (REAL) Voltage value acquired on the resistor to be misured (V).

Almpedance (REAL) Value of input impedance of the acquisition channel (Ohm).

(REAL) Resistor value (Ohm).

Examples

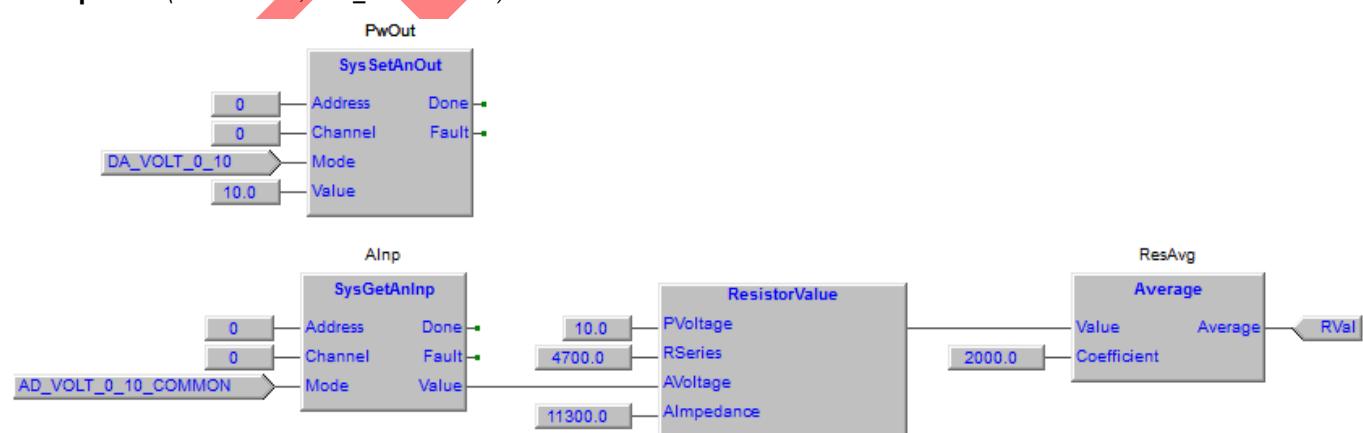
It's calculated, the ohmic value of the resistor connected in a resistor divider with a 4,7KOhm. The divider is supplied with a voltage of 10 Volt. The value is acquired through an analog input module. The acquired value is then filtered with a FB **Average** for having a stable value.



Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	PwOut	SysSetAnOut	Auto	No	Analog output FB
2	Alnp	SysGetAnInp	Auto	No	Analog input FB
3	ResAvg	Average	Auto	No	Resistor value average
4	RVal	REAL	Auto	No	Resistor value (Ohm)

Esempio LD (PTP114A640, FBD_ResistorValue)



7.14 DLMS protocol, or IEC 62056-21

With the new DLMS (Device Language Message Specification) standard protocol, the communication with the metering systems is vastly simplified. This protocol has an **object-oriented** structure that makes it possible to read in the exact same way, data from counters of different manufacturers.

IEC 62056-21 or IEC 61107 is an international standard that describes the DLMS protocol for reading by a computer data from counters tariff of electricity, water and gas.

The protocol includes a step of **Sign-On** with the counter during which you must provide an access code (usually the serial number of the counter), and the counter output provides a password that can be used to encrypt the data.

After this step you can request to counter the value of its records by using the OBIS identification codes composed by 5 characters (IEC 62056-61).

For SlimLine family we have developed a special function block that automates all operations. It is necessary to pass the serial number of the meter and the **OBIS** code of the register to read. The function block performs the sign on on counter and returns the register value indicated.

Interface with the counter

To interface with the meter you can use a special optical coupler that supports the reading of the meter box and connects to a serial port of the SlimLine.

Or in the case of counters with the RS485 port, you can connect it directly to the RS485 port of the SlimLine.

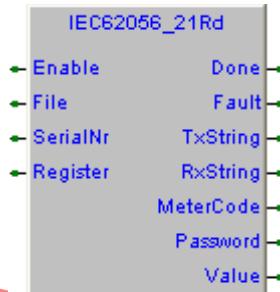


7.14.1 IEC62056_21Rd, IEC62056-21 protocol read

Type	Library
FB	eLLabUtyLib_C030

This function block performs the management of read of registers from metering systems using the protocol IEC62056-21. This function block is protected and require a code to unlock it (see [functions and function blocks protection](#)). It is possible to use it freely in test mode for 30 min.

You must define **SerialNr** with the serial number of the counter (used as a key to reading) and **Register** with the register address to read following the OBIS rules. If the reading is successful, the **Done** output is activated and the variables **MeterCode**, **Password** and **Value** are set with the data read from the counter.



Enable (BOOL)	Enable the read from counter.
File (FILEP)	Stream returned by Sysfopen function.
SerialNr (STRING[16])	Counter serial number. It is used as access key.
Register (STRING[16])	Address of register to read following the OBIS rules.
Done (BOOL)	Active for a program loop at the end of acquisition.
Fault (BOOL)	Active for a program loop if an error during acquisition.
TxString (STRING[32])	Contains the command string sent to the counter. Can be used in debugging to verify the commands sent.
RxString (STRING[32])	Contains the answer string received from the counter. Can be used in debugging to verify the strings received.
MeterCode (STRING[32])	Contains the counter code string received during the Sign-on phase.
Password (UDINT)	Contains the password received during the Sign-on phase.
Value (STRING[32])	Contains the string with the value of the required register.

Error codes

If an error occurs, the **Fault** output is activated and [SysGetLastError](#) can detect the error code.

- 10016010 **File** value not defined.
- 10016020 Protected FB. The available time for demo mode is over.
- 10016050 Execution timeout.
- 10016070 Error on case management.
- 10016100~1 Error receiving the counter type.
- 10016110~2 Error receiving the counter password.
- 10016120~2 Error receiving the counter parameter.
- 10016200 Overflow receiving string from counter.

Examples

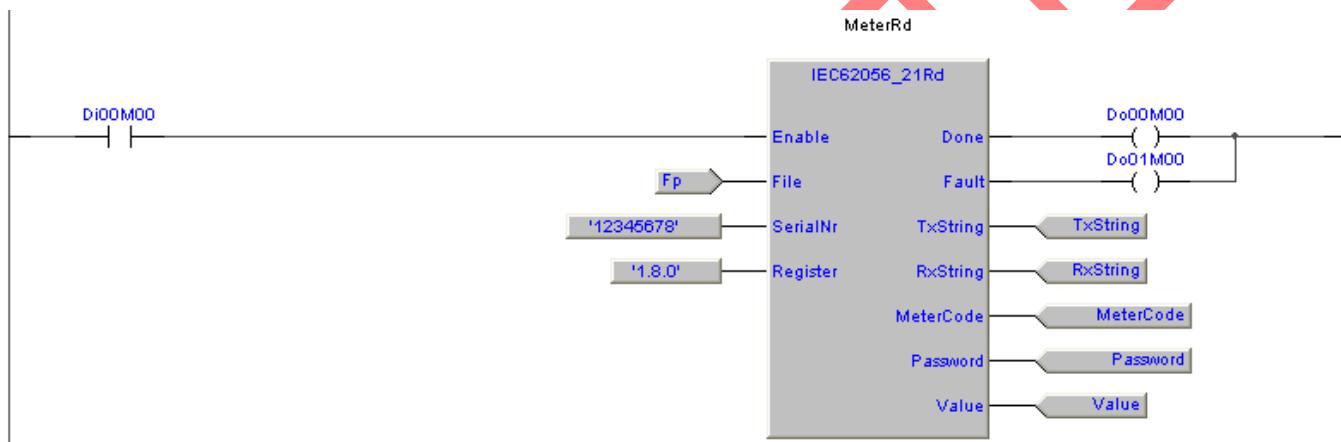
There is a sample program Ptp122*000 that reads 3 records from an energy meter. Here we give an example for reading a single register.

Upon activation of the **Di00M00** digital input, it is executed the read of register **1.8.0** (Power in kW). If the reading is successful, the **Di01M00** digital output is activated for a program loop. The **MeterCode**, **Password** and **Value** variables will be set with values read from the counter.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Fp	FILEP	Auto	No	0	..	File pointer
2	MeterRd	IEC62056_21	Auto	No	0	..	IEC62056_21Rd FB data
3	TxString	STRING	Auto	[32]		..	Tx data string
4	RxString	STRING	Auto	[32]		..	Rx data string
5	MeterCode	STRING	Auto	[32]		..	Meter code
6	Password	UDINT	Auto	No	0	..	Meter password
7	Value	STRING	Auto	[32]		..	Variable read value

LD example (PTP114A000, LD_IEC62056_21Rd)



7.15 Functions and FBs for modem management (eModemLib_E000)

Caution! To use the library it must be imported into your project. See the chapter [how to import libraries](#).

The functions and function blocks for the management of a modem, use a GSM modem connected to an I/O terminal (typically a serial port is used). In the modem must be inserted a SIM card **without PIN code protection**.

In the following description we refer to the following general definitions.

Phone number

The phone number is a string of 10 to 16 numeric characters long with the following format:

International country code without leading zero (eg +39 for Italy, +49 for Germany, +44 for the UK etc..)

Mobile operator code (eg 338, 320, 347, etc..)

Phone number (eg 7589951)

Example: +393337589951, +3933812345, +49172123456

SMS message

An SMS message can be up to 160 characters long with the following set:

A...Z a...z, 0...9, white space, avoid all the other characters.

DEPRECATED

7.15.1 ModemCore_v3, modem core management

Type	Library
FB	eModemLib_E000

This function block manages a modem connected to the I/O device defined in **File**. This function block is protected and require a code to unlock it (see [functions and function blocks protection](#)). It is possible to use it freely in test mode for 30 min.

The FB manages the dialogue with the modem, it performs the initialization and monitors the modem status, see if the modem is connected to the GSM network **operator** and returns the operator and the **Rssi** signal level. If the modem is disconnected from the net, it provides to its reconnection automatically. The FB returns a **ModemID** to be passed to the associated FB (Example send SMS, receive SMS, etc..). The **SpyOn** input allows to spy the FB working.

The **Done** output is activated if the modem is correctly initialized, while the **Fault** output is activated for a program loop in case of error.

There is a **PowerOn** command to manage the modem's power. In this way, the FB can power off and then on the modem if it finds a problem on it.

Upon receiving a telephone call, the CLIP of the caller is detected and returned in **CLIPNumber** output. At the same time every ring of the phone, the **CLIPRxd** output is activated for a program loop. On **IPAddress** it's returned the assigned IP on a GPRS connection.



Enable (BOOL)	Enable function block. In this way the modem will be managed.
AutoHangUp (BOOL)	Automatically hangs up the moden on a call receive, the FB returns the CLIP
SpyOn (BOOL)	Active allows to spy the FB working.
File (FILEP)	Stream returned by Sysfopen function.
APN (STRING[32])	APN definition for the GPRS connection.
DNS (STRING[16])	DNS server IP address.
Enabled (BOOL)	Function block enabled.
Done (BOOL)	Modem correctly initialized and working.
Fault (BOOL)	Active for a program loop if management error.
PowerOn (BOOL)	Command to use to power on and off modem.
CLIPRxd (BOOL)	Active for a program loop every CLIP reception (Tipically every RING).
ModemID (UDINT)	Modem ID to pass to linked FBs (Example ModemSMSend , ModemSMSReceive , etc.).
CnStatus (USINT)	Connection status. 0: Not registered. 1: Registered to home network. 2: Not registered, but ME is currently searching for a new operator. 3: Registration denied. 4: Unknown (not used). 5: Registered, roaming.
Operator (STRING[16])	Contain the string with the network operator.
Rssi (USINT)	Value of Received signal strength indication.
CLIPNumber (STRING[16])	Contains the string with the received CLIP number.
IPAddress (STRING[16])	IP address assigned by the network on a GPRS connection.

Spy trigger

If **SpyOn** is active the [**SysSpyData**](#) function is executed this allows to spy the FB operations. There are various levels of triggers.

TFlags Description

- 16#00000001 'Rx' String received from modem
- 16#00000002 'R>' String received from modem is too long
- 16#00000004 'Tx' String transmitted to modem
- 16#00000010 'Mr' SMS received
- 16#01000000 '--' Modem operation labels

Error codes

If an error occurs, the **Fault** output is activated and [**SysGetLastError**](#) can detect the error code.

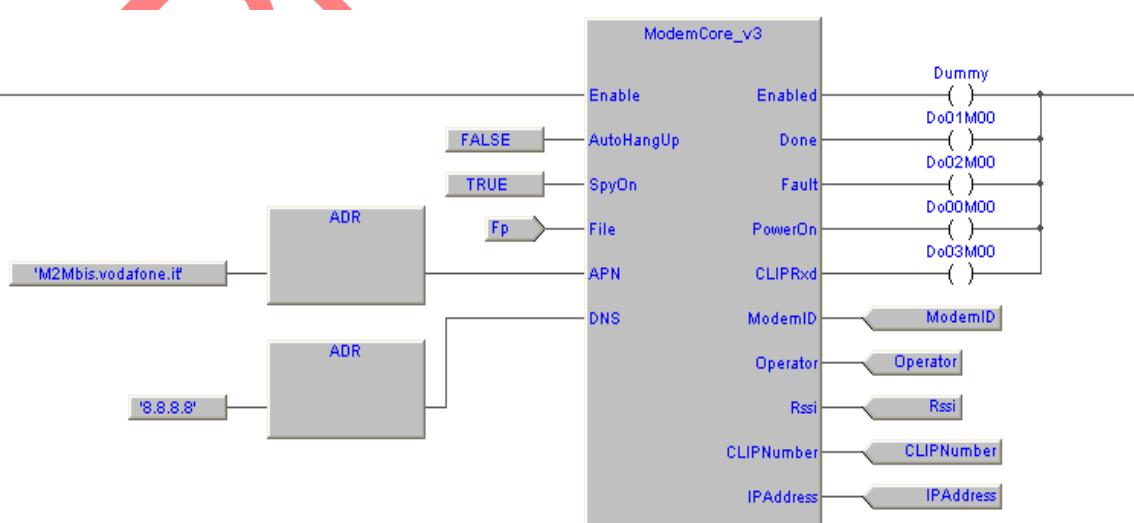
Code Description

- 10002010 **File** value not defined.
- 10002020 Protected FB. The available time for demo mode is over.
- 10002050 Execution timeout.
- 10002080 Management case mismatch.
- 10002100 Wrong Rx data from modem.
- 10002200~3 Error receiving CLIP.
- 10002300~3 Error in the power on sequence.
- 10002400~12 Error in the sequences for modem controlling.
- 10002350~7 Error receiving the SMS message.

Examples

In the example is managed a modem connected to the terminal I/O defined in the **Fp** variable. For definitions of variables and a better understanding of the operation, see the examples below.

LD example



7.15.2 ModemSMSReceive, receive a SMS message

Type	Library
FB	eModemLib_E000

This function block performs the reception of a SMS message. It connects to the **ModemCore** function block using the **ModemID** input.

When a SMS message is received, the **Done** output will be activated for a program loop. On the **SMSText** output is returned the message text received. On the **CLIPNumber** output of **ModemCore** function block, is returned the phone number from which the message was received. The text of the received message is present in output until the reception of another message.



Enable (BOOL) Enables the reception of SMS messages.

ModemID (UDINT) Modem ID supplied as output from [ModemCore](#).

Done (BOOL) Active for a program loop when a SMS is received.

Fault (BOOL) Active for a program loop if an error detected.

Text (STRING[160]) Received SMS message text.

Error codes

If an error occurs, the **Fault** output is activated and [SysGetLastError](#) can detect the error code.

10003010 **ModemID** not defined.

10003020 **ModemID** not defined.

Example

The example handles the reception of an SMS message from the **ModemID** modem. For definitions of variables and a better understanding of the operation, see the examples below.

LD example (Ptp118b100, ReceiveSMSMessage)



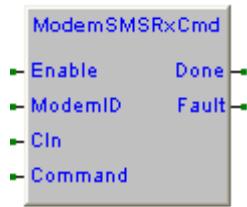
7.15.3 ModemSMSRxCmd_v1, receive a SMS command

Type	Library
FB	eModemLib_E000

This function block performs the reception of a command via a SMS message. It connects to the **ModemCore** function block using the **ModemID** input.

When a SMS message is received, the string defined in **Command** will be searched in the received message. If found, the **Done** output will be activated for a program loop. In the **CLIPNumber** output of the **ModemCore** FB, it is returned the phone number from which the message was received.

Activating the **Cin** input, the search of **Command** string in the received string will be made without considering the case (upper case / lower case) character.



Enable (BOOL) Enables the reception of SMS command.

ModemID (UDINT) Modem ID supplied as output from **ModemCore**.

Cin (BOOL) If active, the search of **Command** string, will be made without considering the case (upper case/lower case) characters.

Command (@USINT) Pointer to the command text to execute.

Done (BOOL) Active for a program loop when a SMS containing the **Command** text is received.

Fault (BOOL) Active for a program loop if there is an error.

Error codes

If an error occurs, the **Fault** output is activated and [SysGetLastError](#) can detect the error code.

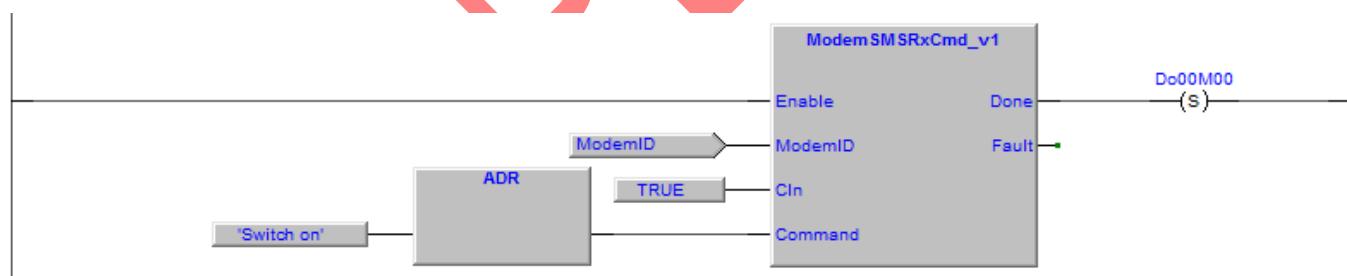
10004010 **ModemID** not defined.

10004020 **ModemID** not defined.

Examples

The example handles the reception of an SMS message from the **ModemID** modem. For definitions of variables and a better understanding of the operation, see the examples below.

LD example ([Ptp118b100, ReceiveSMSCommand](#))



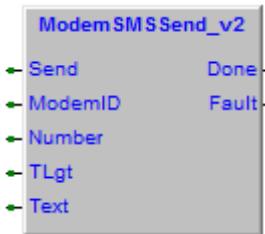
7.15.4 ModemSMSend_v2, send a SMS message

Type	Library
FB	eModemLib_E000

This function block performs the sending of an SMS message. It connects to the **ModemCore** function block using the **ModemID** input.

On the rising edge of **Send** input the send of message it's booked. As soon as possible, the text message defined in **Text** will be sent to the number defined in **Number**. After sending, the **Done** output will be enabled for a program loop.

The **Tlgt** parameter defines the number of characters of **Text** must be sent. If not defined or set to "0" the entire **Text** string will be send.



Send (BOOL)	On the rising edge, it force sending of message. Warning! The message will be sent as soon as modem will be free to send.
ModemID (UDINT)	Modem ID supplied as output from ModemCore .
Number (@USINT)	Pointer to phone number to which to send message.
TLgt (UDINT)	Number of characters of Text must be sent. If "0" the entire Text string is sent.
Text (@USINT)	Pointer to message text.
Done (BOOL)	Active for a program loop when the message was sent.
Fault (BOOL)	Active for a program loop if there is an error.

Error codes

If an error occurs, the **Fault** output is activated and [SysGetLastError](#) can detect the error code.

10005010 **ModemID** not defined.

10005020 **ModemID** not defined.

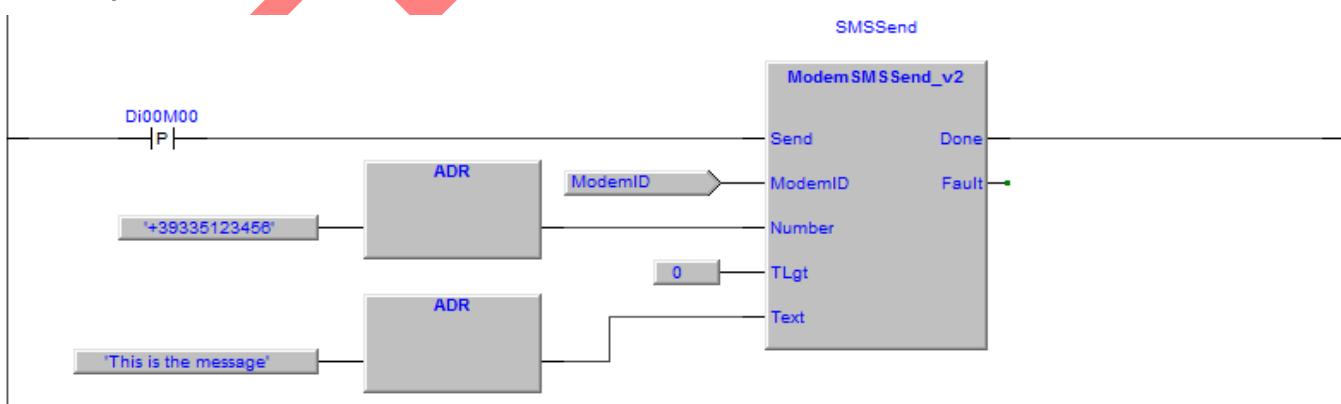
10005080 Management case mismatch.

10005100~3 SMS send error.

Examples

The example handles the sending of an SMS message on modem defined by **ModemID** variable. For definitions of variables and a better understanding of the operation, see the examples below.

LD example ([Ptp118b100, SendSMSMessage](#))



Sending a message to multiple numbers

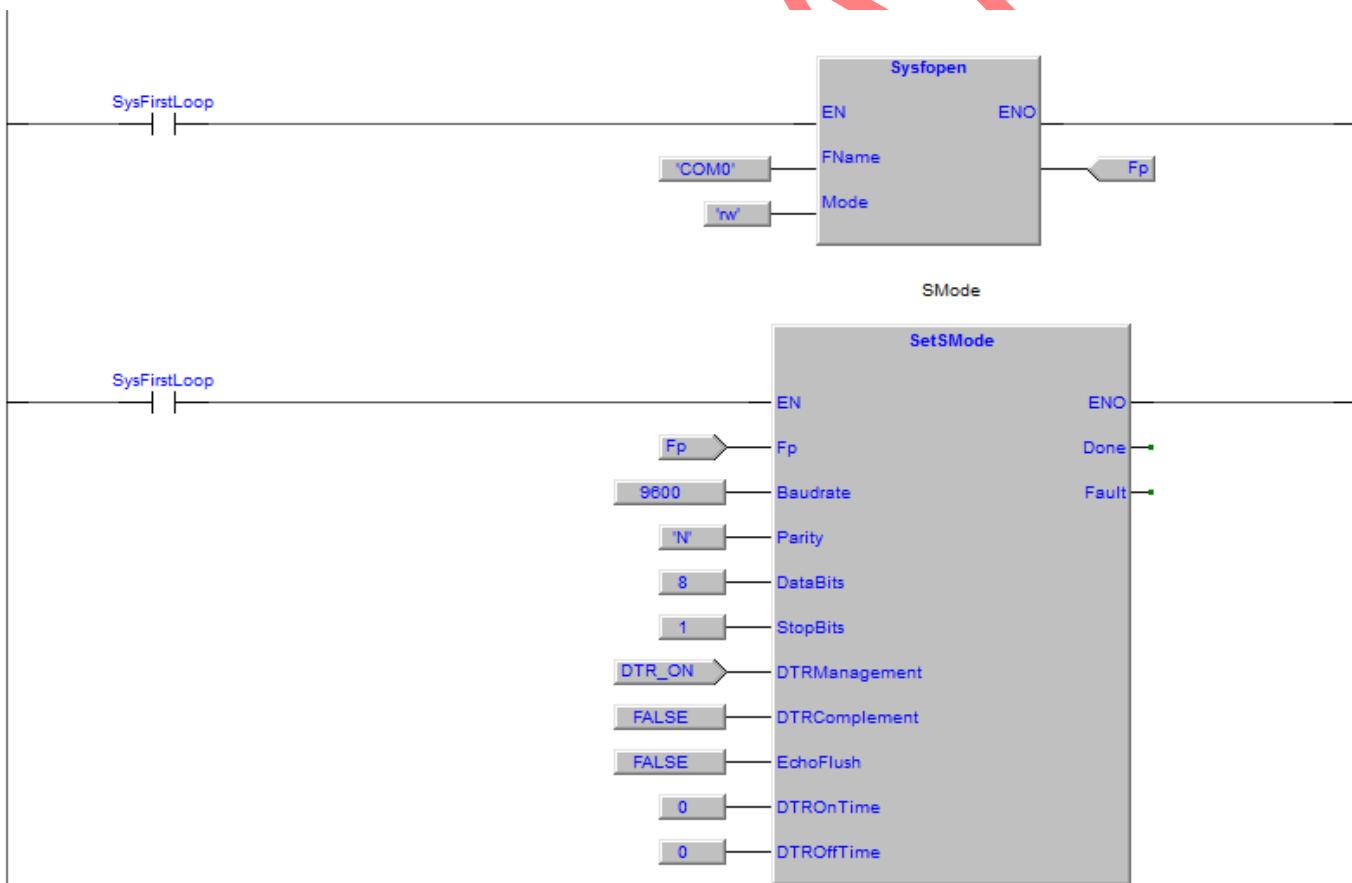
Many times it happens you have to send on an event (Example a digital input) multiple SMS messages, each with their own text to different phone numbers. The **SMSSend** function block allows the sending on the same event of multiple messages, each message is identified by the phone number and the text message.

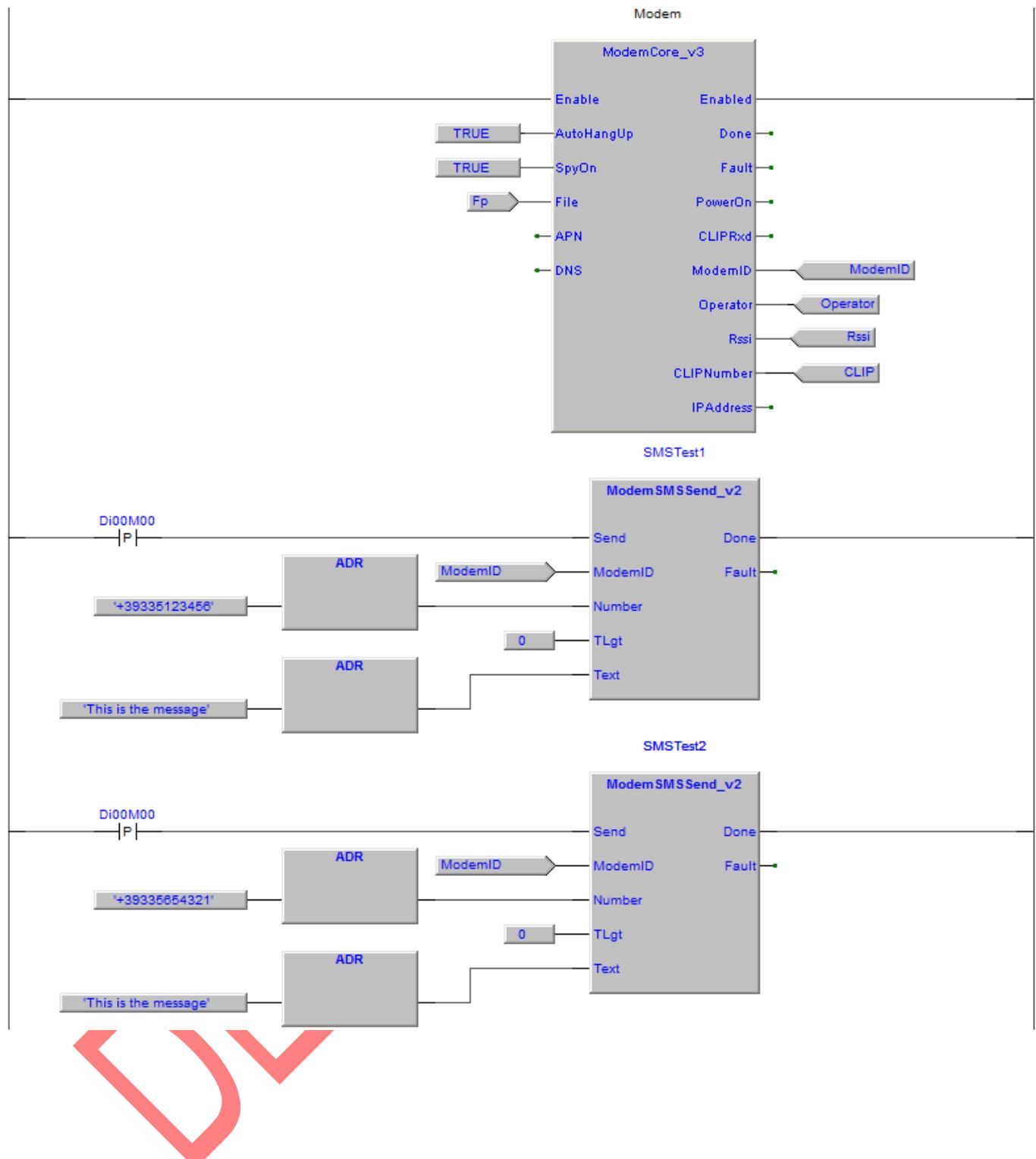
In the example is sent a message to multiple phone numbers. By activating the digital input **Di00M00** will be sent two messages to two different phone numbers. And it's possible to add more branches to the function block **SMSSend** each with its own message and telephone number, all branches are enabled by the **Di00M00**.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Rssi	USINT	Auto	No	..		RSSI value
2	ModemID	UDINT	Auto	No	..		Modem ID
3	CLIP	STRING	Auto	[16]	..		CLIP received number
4	Operator	STRING	Auto	[16]	..		Operator value
5	Fp	FILEP	Auto	No	..		Serial pointer
6	Modem	ModemCore_v3	Auto	No	..		Modem core FB
7	SMSTest1	ModemSMSSend_v2	Auto	No	..		Modem SMS send FB
8	SMSTest2	ModemSMSSend_v2	Auto	No	..		Modem SMS send FB
9	SMode	SetSMode	Auto	No	..		Serial mode FB

LD example (Ptp118b100, MultipleSMSSend)



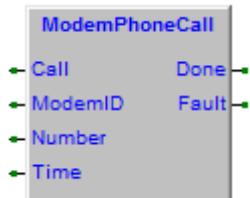


7.15.5 ModemPhoneCall_v1, executes a phone call

Type	Library
FB	eModemLib_E000

This function block executes a call to the telephone number defined. It connects to the **ModemCore** function block using the **ModemID** input.

On the rising edge of **Call** input the phone call it's booked. As soon as possible, the phone call to the number defined in **Number** it's been done. If there are no problems the **Done** output will be activated for a program loop.



Call (BOOL)	On the rising edge, it force the phone call. Warning! The call will be executed as soon as modem will be free.
ModemID (UDINT)	Modem ID supplied as output from ModemCore .
Number (@USINT)	Pointer to the phone number to call.
Time (UINT)	Phone call delay (Sec).
Done (BOOL)	Active for a program loop when the call is executed.
Fault (BOOL)	Active for a program loop if there is an error.

Error codes

If an error occurs, the **Fault** output is activated and [SysGetLastError](#) can detect the error code.

10037010 **ModemID** not defined.

10037020 **ModemID** not correct.

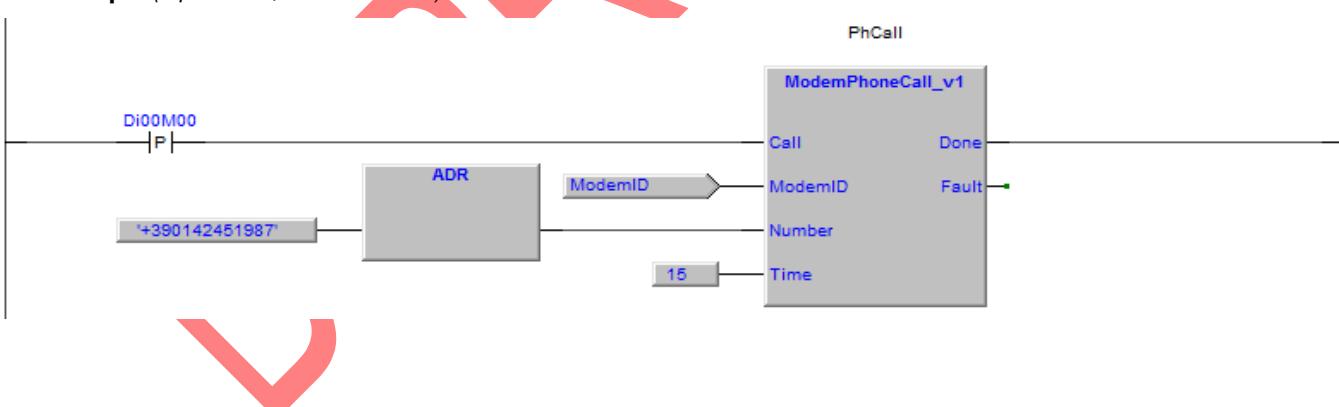
10037080 Management case mismatch.

10037100~3 Phone call sequences error.

Example

In the example a call at the number defined it's executed. After 15 seconds the call is ended.

LD example (Ptp118b100, MakePhoneCall)



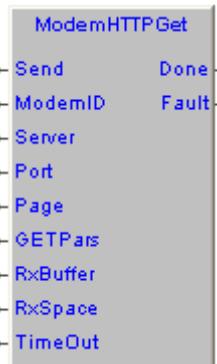
7.15.6 ModemHTTPGet, executes a HTTP Get request

Type	Library
FB	eModemLib_E000

This function block executes an HTTP request by inserting in line the GET parameters. It connects to the **ModemCore** function block using the **ModemID** input.

On the rising edge of **Send** the modem connects to the HTPP **Server** on defined **Port**, and requests the defined **Page**. The page is requested with GET parameters defined in the buffer pointed to by **GETPars**. The page received from the server is transferred to the buffer pointed by **RxBuffer**, the data received that exceed the size of the buffer **RxSpace** are discarded.

After the requested page has been received the **Done** output is activated for a loop, if the page is not returned within the time defined in **TimeOut** execution terminates with an error.



Send (BOOL)

On the rising edge, it force the page request. **Warning!** The request will be executed as soon as modem will be free.

ModemID (UDINT)

Modem ID supplied as output from [ModemCore](#).

Server (@USINT)

Pointer to HTTP server definition string.

Port (UINT)

Port number on which connect.

Page (@USINT)

Pointer to page definition string.

GETPars (@USINT)

Pointer to GET parameters definition string (Up to 512 characters).

RxBuffer (@USINT)

Pointer to page reception buffer (Up to 360 characters).

RXSpace (UINT)

Page reception space definition.

TimeOut (UINT)

Maximum time waiting page (mS).

Done (BOOL)

Active for a program loop when the page is been received.

Fault (BOOL)

Active for a program loop if there is an error.

Error codes

If an error occurs, the **Fault** output is activated and [SysGetLastError](#) can detect the error code.

10042010 **ModemID** not defined.

10042020 **ModemID** not correct.

10042080 Management case mismatch.

10042100~8 HTTP page request sequences error.

10042200 GET parameters string too long.

Examples

The typical use of this FB is to connect to an HTTP server where script pages (ASP, PHP, Python, etc.) are executed requesting the page providing the GET parameters on-line to the call. The script of the page can perform operations with data passed in line and return values as a result. Here is a simple example of a page using PHP script on the server allows you to send an Email.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Pulse	BOOL	Auto	No	..		Auxiliary pulse
2	Result	BOOL	Auto	No	..		Result flag
3	ABf	INT	Auto	No	..		Auxiliary buffer
4	RxD	STRING	Auto	[32]	..		HTTP Rx data
5	TxD	STRING	Auto	[128]	..		HTTP Tx data
6	Mdm	ModemCore_v3	Auto	No	..		Modem management
7	HTTP	ModemHTTPGet	Auto	No	..		HTTP Get
8	Sm	SYSSERIALMODE	Auto	No	..		Serial mode

ST Example (Ptp118b100, EmailWithHTTPGet)

```

(* ****)
(* PROGRAM "EmailWithHTTPGet")
(* ****)
(* This program sends an Email by contacting a PHP script.
(* -----)

(* -----
(* INITIALIZATION
(* -----
(* General initializations. *)

IF (SysFirstLoop) THEN
    (* Opening the serial port on which the modem is connected. *)
    Mdm.File:=Sysfopen('COM0', 'rw'); (* COM port file pointer *)
    (* Set the serial communication mode. *)
    ABf:=SysGetSerialMode(ADR(Sm), Mdm.File);
    Sm.Baudrate:=115200; (* Baudrate *)
    Sm.Parity:='N'; (* Parity *)
    Sm.DataBits:=8; (* Data bits *)
    Sm.StopBits:=1; (* Stop bits *)
    Sm.DTRManagement:=DTR_OFF; (* DTR management *)
    Sm.DTRComplement:=FALSE; (* DTR complement *)
    Sm.EchoFlush:=TRUE; (* Flush the echo characters *)
    ABf:=SysSetSerialMode(ADR(Sm), Mdm.File);

    (* Modem parameters. Please change them according your requirements. *)
    Mdm.AutoHangUp:=TRUE; (* Auto hangup *)
    Mdm.APN:=ADR('M2Mbis.vodafone.it'); (* APN definition *)
    Mdm.DNS:=ADR('8.8.8.8'); (* DNS definition *)

    (* HTTP parameters. Please change them according your requirements. *)
    HTTP.Server:=ADR('www.MyServer.com'); (* HTTP server *)
    HTTP.Port:=80; (* HTTP port *)
    HTTP.Page:=ADR('eMailSend.php'); (* Page name *)
    HTTP.TimeOut:=5000; (* Connection timeout (mS) *)
    HTTP.GETPars:=ADR(TxD); (* GET parameters pointer *)
    HTTP.RxBuffer:=ADR(RxD); (* Rx buffer pointer *)
    HTTP.RxSpace:=SIZEOF(RxD); (* Rx buffer space *)
END_IF;

(* -----
(* FBs MANAGEMENT
(* -----
(* Manage the modem core. *)

```

```

Mdm(Enable:=TRUE); (* Modem core management *)
HTTP.ModemID:=Mdm.ModemID; (* Modem ID *)

(* Manage the HTTP connection. *)

HTTP();
HTTP.Send:=FALSE;

(* ----- *)
(* GET THE HTTP PAGE AND SEND THE EMAIL *)
(* ----- *)
(* On digital input activation the HTTP page is requested. *)

IF (Di00M00 <> Pulse) THEN
    Pulse:=Di00M00; (* Auxiliary pulse *)

    IF (Pulse) THEN
        ABf:=SysVarsnprintf(ADR(TxD), 128, 'To=%s', STRING_TYPE, ADR('sbertana@elsist.it'));
        ABf:=SysVarsnprintf(ADR(TxD)+LEN(TxD), 128-LEN(TxD), '$26Subject=%s', STRING_TYPE,
ADR('Subject'));
        ABf:=SysVarsnprintf(ADR(TxD)+LEN(TxD), 128-LEN(TxD), '$26Text=%s', STRING_TYPE, ADR('Text'));
        HTTP.Send:=TRUE;
    END_IF;
END_IF;

(* Check the HTTP request answer. *)

IF (HTTP.Done) THEN
    IF (FIND(RxD, 'Ok') = 0) THEN Result:=FALSE; ELSE Result:=TRUE; END_IF;
END_IF;

(* [End of file] *)

```

Let's explain how the program works, on rising edge of Di00M00 it's made the connection to the server www.MyServer.com and it's requested the eMailSend.php page. In line to the call of the page are placed the GET fields **To=sbertana@elsist.it&Subject=Subject&Text=Text**. Note that the "&" character has been replaced by "\$26".

The **eMailSend.php** page on server contains a PHP script like this:

```

<?php
$Headers="MIME-Version: 1.0\n";
$Headers.="Content-type: text/html; charset=iso-8859-1\n";
$Headers.="To: ".$_REQUEST['To']."\n";
$Headers.="From: \n";
mail($_REQUEST['To'], $_REQUEST['Subject'], $_REQUEST['Text'], $Headers);
echo "Ok";
exit();
?>

```

As you can see the script sends a mail by using the mail command whose fields are defined by the parameters set in the GET request. The statement **\$_REQUEST['To']** contains the text defined in **To=**, **\$_REQUEST['Subject']** contains the text defined in **Subject=**, and so on. As you can see it is possible to pass to the script all the values that you want.

The return value from the script, defined with the echo statement, will be transferred into the **RxD** buffer of our program and then you can work on it with the string handling instructions.

7.16 Functions and FBs for One-Wire management (ePLC1WireLib_C000)

Caution! To use the library it must be imported into your project. See the chapter [how to import libraries](#).

The 1 Wire® is a standard protocol, based, as indicated by the same name, on a single wire of communication. It supports many devices and sensors commonly used in industrial and domestic automation.

The devices are interconnected by only two wires: one for ground and one for signal and power. On these two wires can be connected all the devices on the network by choosing the physical layout required.

The 1 Wire® protocol is equipped with all modes of communication that allow you to achieve a high data transfer and an intrinsic safety on their validity. This is due to unique addressing techniques, polynomial CRC control, many verification commands and complex management algorithms.

On SlimLine systems you can connect a serial/One-Wire converter to serial port COM0 or COM1 on the CPU module, or you can use any serial port present on an extension module in order to have one or more One-Wire bus.

The **sOWireMng** function block manages the converter allowing management of devices connected to it.



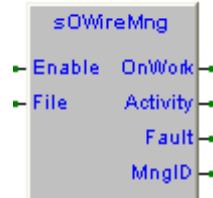
DEPREC'D

7.16.1 sOWireMng, One-Wire management

Type	Library
FB	ePLC1WireLib_C000

This function block manages the Serial/One-Wire converter connected to the I/O device defined in **File**. The FB handles the initialization and management of the converter.

The **OnWork** output is activated if **Enable** is active, the fault output is activated in case of mismanagement and remains active until the error persists. Activity output is active during the One-Wire bus activity. The FB returns a **MngID** to be passed to the connected FBs.



Enable (BOOL)	Enables the management of Serial/One-Wire converter.
File (FILEP)	Stream returned by Sysopen function.
OnWork (BOOL)	Function block activated ed on work.
Activity (BOOL)	Active on One-Wire bus activity.
Fault (BOOL)	Active on One-Wire bus error.
MngID (UDINT)	One-Wire ID to pass to the linked FBs (Example sOWRdIdentifier , sOWRdTemeprature , etc.).

Error codes

If an error occurs, the **Fault** output is activated and [SysGetLastError](#) can detect the error code.

- 10008010 **File** value not defined.
- 10008200 Timeout on sending command string.
- 10008300~9 Error configuring converter.
- 10008400~1 Error verifying configuration.
- 10008500~2 Error in sequences for addressing the One-Wire device.
- 10008600 Answer error to reset pulse command.
- 10008601 One-Wire bus shortcircuit.
- 10008602 Error on One-Wire devices.
- 10008603 No One-Wire devices on the bus.

7.16.2 sOWRdIdentifier, One-Wire read ROM identifier

Type	Library
FB	ePLC1WireLib_C000

This function block reads the identification code of a One-Wire device. It connects to the **sOWireMng** function block using the **MngID** input.

Activating the **Enable** command, is executed a read of the ID from the ROM device connected to the One-Wire bus. **Warning! You must have one only device connected to the bus**. After reading the code **Done** is set. If the reading is successful the **Ok** output is set for a program loop and the 8 bytes code read is transferred into the array addressed by **IDCode**. Disabling **Enable** the **Done** and **Fault** outputs are reset, to execute a new reading the **Enable** input must be reactivated.



Enable (BOOL)	Enable function block.
MngID (UDINT)	One-Wire ID supplied as output from OWireMng .
IDCode (@USINT)	Pointer to array where to copy the read ROM ID. It must be at least 8 byte long.
Done (BOOL)	Activated at the end of the ROM ID reading.
Ok (BOOL)	Active for a program loop if ROM ID reading has been correctly done.
Fault (BOOL)	Active for a program loop if there is an error.

Error codes

If an error occurs, the **Fault** output is activated and [SysGetLastError](#) can detect the error code.

- 10009010 **MngID** not defined.
- 10009020 **MngID** not defined.
- 10009030 **IDCode** not defined.
- 10009100 FB **OWireMng**, Serial/One-Wire converter management busy.
- 10009200~2 Error on One-Wire sequence reading ID.

Examples

Here's a simple example of a program for the management of iButton devices for personal recognition. Inserting the TAG in the reader, the ROM identifier is read and the value acquired is transferred to the **ROMID** array of 6 bytes.

Cyclically is performed an acquisition, if a TAG is inserted in the reader, the **TAGInserted** flag is activated.

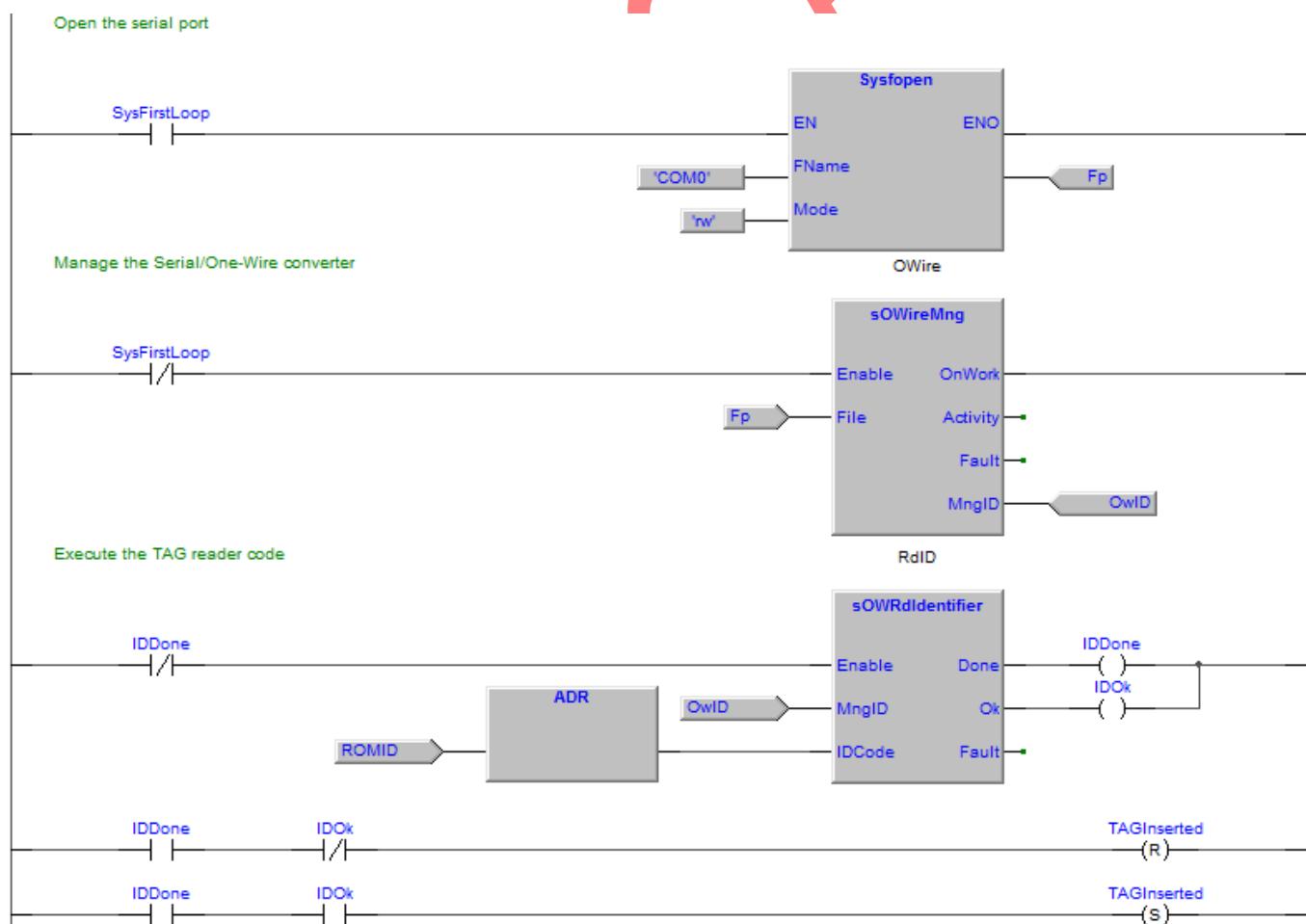
For simplicity, the program does not run any control to the ID read, but in an access control system for example it is possible to identify the person and enable or not the access.



Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	IDDone	BOOL	Auto	No	FALSE	..	Read ID code done
2	IDOk	BOOL	Auto	No	FALSE	..	ID code read
3	TAGInserted	BOOL	Auto	No	FALSE	..	TAG inserted on reader
4	ROMID	BYTE	Auto	[0..7]	8(0)	..	ROM ID code
5	OwID	UDINT	Auto	No	0	..	One-Wire ID
6	Fp	FILEP	Auto	No	0	..	File pointer
7	OWire	sOWireMng	Auto	No	0	..	FB One Wire management
8	RdID	sOWRdIdentifier	Auto	No	0	..	FB One Wire read ID

LD example (PTP120A200, LD_TAGReader)



7.16.3 sOWRdTemperature, One-Wire read temperature

Type	Library
FB	ePLC1WireLib_C000

This function block performs the acquisition of a One-Wire temperature sensor (Maxim DS18B20). It connects to the **sOWireMng** function block using the **MngID** input.

By activating **Enable** command, is executed a read of temperature value from the device connected to the One-Wire bus. Executed the acquisition the **Done** output become active, if the value has been correctly read the **Ok** output is set for a program loop and the acquired value is reported on **Temperature** output.



The **Fault** output is activated in case of error. Disabling **Enable** the **Done** and **Fault** outputs are reset, to execute a new reading the **Enable** input must be reactivated.

If the One-Wire bus is connected to a single device, you can avoid setting **IDCode** or left it **0**. However, if the One-Wire bus have multiple parallel devices, in **IDCode** you must define the address of the array of 8 bytes contains the ROM ID of the device you want to acquire.

Enable (BOOL)	Enables function block.
MngID (UDINT)	One-Wire ID supplied as output from OWireMng .
IDCode (@USINT)	Pointer to array where to copy the read ROM ID. It must be at least 8 byte long.
Done (BOOL)	Activated at the end of the temperature reading.
Ok (BOOL)	Active for a program loop if temperature reading has been correctly done.
Fault (BOOL)	Active for a program loop if there is an error.
Temperature (REAL)	Acquired temperature value (°C). Range from -55 (°C) to +125 (°C). Precision ±0.5 (°C) from -10 (°C) to +85 (°C). Resolution 0.0625 (°C).

Error codes

If an error occurs, the **Fault** output is activated and [SysGetLastError](#) can detect the error code.

- 10010010 **OWireID** not defined.
- 10010020 **OWireID** not defined.
- 10010100 FB **OWireMng**, Serial/One-Wire converter management busy.
- 10010200~5 Error on read temperature sequence.

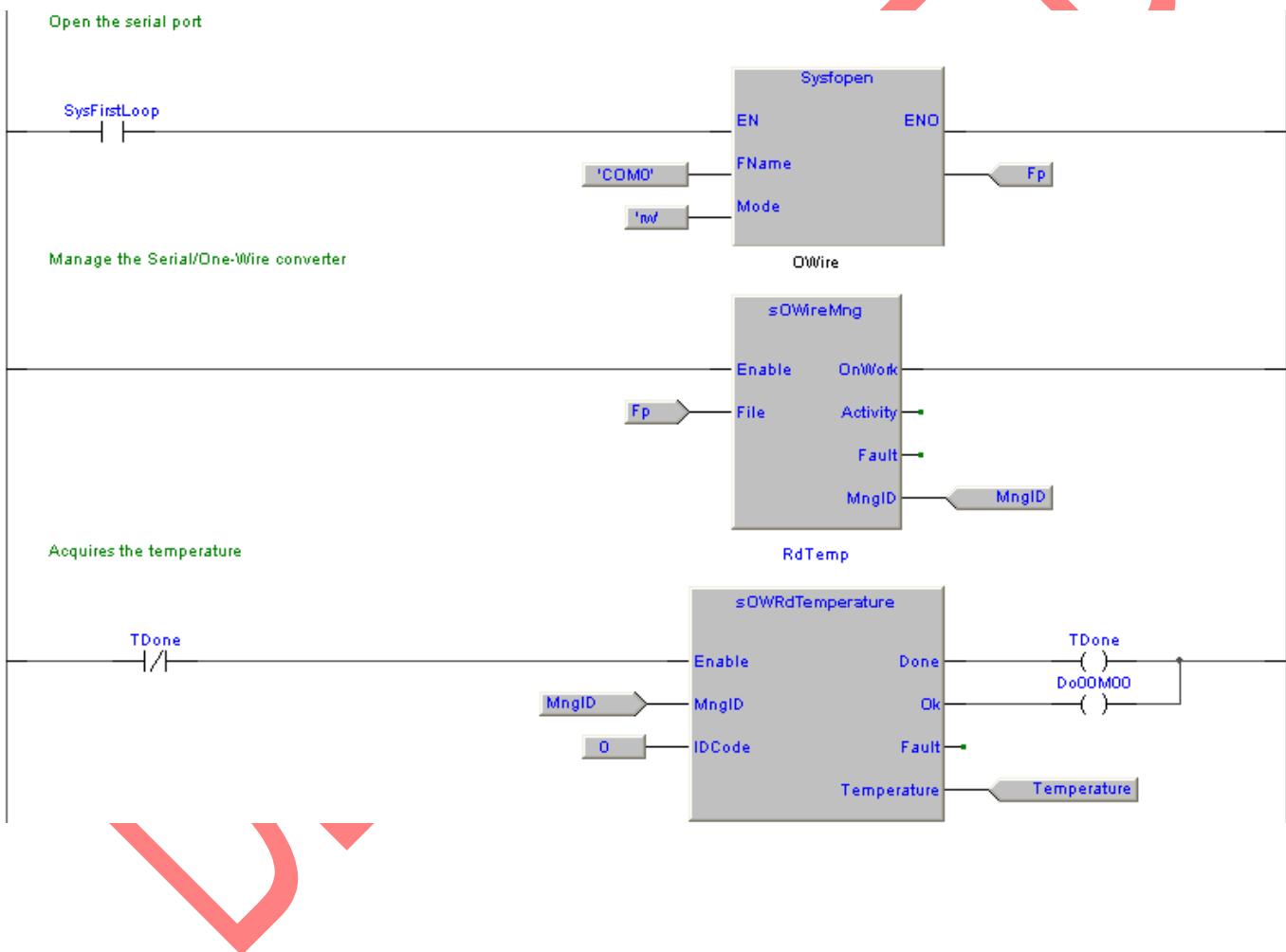
Examples

It is executed the read of temperature from a One-Wire device. Not being defined the **IDCode**, any device on the One-Wire bus will be acquired. **Warning! There must be a single device on the bus**. If the execution is correct, the **Do00M00** output is activated and the acquired value is transferred into the **Temperature** variable.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Fp	FILEP	Auto	No	0	..	File pointer
2	MngID	UDINT	Auto	No	0	..	One-Wire management ID
3	Temperature	REAL	Auto	No	0	..	Temperature (°C)
4	RdTemp	sOWRdTemperature	Auto	No	0	..	Read temperature FB
5	TDone	BOOL	Auto	No	FALSE	..	Read temperature done
6	OWire	sOWireMng	Auto	No	0	..	One Wire management FB

LD example (PTP120A200, LD_SingleTempSkipROM)



7.16.4 sOWRdHumidity, One-Wire read humidity

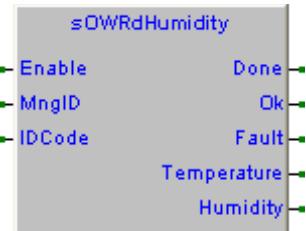
Type	Library
FB	ePLC1WireLib_C000

This function block performs the acquisition of a One-Wire temperature and humidity sensor (Maxim DS2438). It connects to the **OWireMng** function block using the **MngID** input.

By activating **Enable** command, is executed a read of temperature and humidity values from the device connected to the One-Wire bus. Executed the acquisition the **Done** output become active, if the value has been correctly read the **Ok** output is set for a program loop and the acquired values are reported on **Temperature** and **Humidity** outputs.

The **Fault** output is activated in case of error. Disabling **Enable** the **Done** and **Fault** outputs are reset, to execute a new reading the **Enable** input must be reactivated.

If the One-Wire bus is connected to a single device, you can avoid setting **IDCode** or left it **0**. However, if the One-Wire bus have multiple parallel devices, in **IDCode** you must define the address of the array of 8 bytes contains the ROM ID of the device you want to acquire.



Enable (BOOL)	Enables function block.
MngID (UDINT)	One-Wire ID supplied as output from OWireMng .
IDCode (@USINT)	Pointer to array where to copy the read ROM ID. It must be at least 8 byte long.
Done (BOOL)	Activated at the end of the temperature and humidity reading.
Ok (BOOL)	Active for a program loop if temperature and humidity reading has been correctly done.
Fault (BOOL)	Active for a program loop if there is an error.
Temperature (REAL)	Acquired temperature value (°C). Range from -55 (°C) to +125 (°C). Precision ±0.5 (°C) from -10 (°C) to +85 (°C). Resolution 0.03125 (°C).
Humidity (REAL)	Acquired humidity value (RH%).

Error codes

If an error occurs, the **Fault** output is activated and [SysGetLastError](#) can detect the error code.

- 10015010 **OWireID** not defined.
- 10015020 **OWireID** not defined.
- 10015100 FB **OWireMng**, Serial/One-Wire converter management busy.
- 10015200~3 Error on temperature conversion sequences.
- 10015300~8 Error acquiring the power supply voltage of sensor.
- 10015400~8 Errore acquiring the humidity sensor.

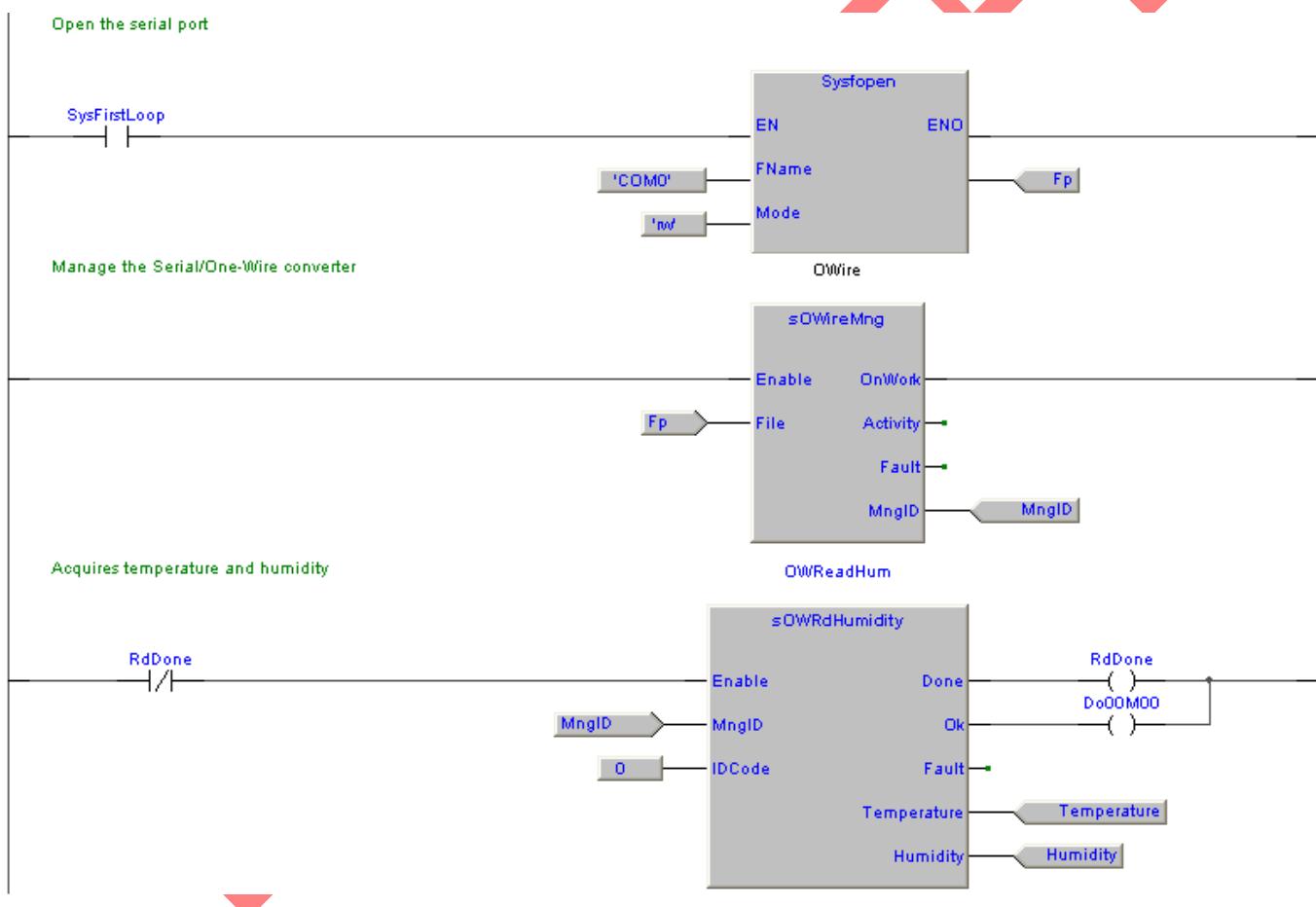
Examples

It is executed the read of temperature and humidity from a One-Wire device. Not being defined the **IDCode**, any device on the One-Wire bus will be acquired. **Warning! There must be a single device on the bus.** If the execution is correct, the **Do00M00** output is activated and the acquired value is transferred into the **Temperature** and **Humidity** variables.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	RdDone	BOOL	Auto	No	FALSE	..	Read humidity/temperature done
2	MngID	UDINT	Auto	No	0	..	One-Wire management ID
3	Humidity	REAL	Auto	No	0	..	Humidity (RH%)
4	Temperature	REAL	Auto	No	0	..	Temperature (°C)
5	Fp	FILEP	Auto	No	0	..	File pointer
6	OWire	sOWireMng	Auto	No	0	..	One-Wire management FB
7	OWReadHum	sOWRdHumicAuto	No	0	FB read humidity data

LD example (PTP120A200, LD_HumiditySkipROM)



7.17 Functions and FBs for networking management

From IEC language, some functions and function blocks are available for networking management.

DEPRECATED

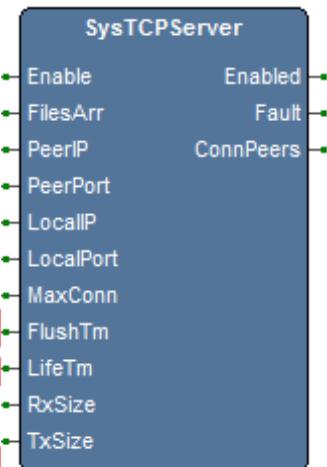
7.17.1 SysTCPServer, accepts TCP/IP connections

Type	Library
FB	XTarget_12_0_0

Questa function block gestisce la comunicazione con protocollo TCP/IP in modalità server. Occorre fornire l'indirizzo di un array di flussi dati streams al parametro **FilesArr**. Occorre definire la porta TCP da porre in ascolto ed il numero di connessioni contemporanee accettate.

Attivando il comando **Enable** il server TCP viene posto in ascolto sulla porta indicata, sulla connessione di un client viene incrementato il numero di **ConnPeers** ed uno degli stream definiti in **FileArr** viene valorizzato ed aperto. Sullo stream aperto è possibile utilizzare le funzioni di TermIO per gestire la comunicazione.

Per modificare i parametri occorre disattivare e poi riabilitare il comando **Enable**.



Enable (BOOL)	Comando abilitazione server.
FilesArr (@FILEP)	Pointer ad array streams di I/O. I vari file streams saranno valorizzati alla connessione dei clients. Occorre definire un numero di streams pari al numero di connessioni contemporanee accettate.
PeerIP (STRING[15])	Range indirizzi IP da cui è accettata la connessione. La connessione è accettata se indirizzo IP del peer in AND con il valore non viene modificato. Default 255.255.255.255 : connessione accettata da tutti gli indirizzi IP.
PeerPort (UINT)	Range porte da cui è accettata la connessione. La connessione è accettata se porta del peer in AND con il valore non viene modificata. Default 65535 : connessione accettata da tutte le porte.
LocalIP (STRING[15])	Indirizzo IP della interfaccia di rete da cui si accettano le connessioni. La connessione è accettata se IP del peer in AND con il valore non viene modificato
LocalPort (UINT)	Numero di porta in ascolto sul server.
MaxConn (USINT)	Numero massimo di connessioni contemporanee accettate dal server. Deve essere uguale al numero di files definiti.
FlushTm (UINT)	Tempo di flush dati, se non sono caricati dati sullo stream dopo il tempo definito i dati presenti vengono automaticamente inviati (mS).
LifeTm (UINT)	Tempo di vita socket, se non sono ricevuti o inviati dati dopo il tempo definito il socket viene automaticamente chiuso (Sec). Se definito tempo "0" il socket non viene mai chiuso.
RxSize (UINT)	Dimensione buffer ricezione dati.
TxSize (UINT)	Dimensione buffer trasmissione dati.
Enabled (BOOL)	Attivo se TCP server correttamente impostato e pronto.
Fault (BOOL)	Attivo se errore gestione.
ConnPeers (FILEP)	Numero di clients connessi al server.

Codici di errore

In caso di errore si attiva l'uscita **Fault**, con [**SysGetLastError**](#) è possibile rilevare il codice di errore.

Codice Descrizione

- 9942005 Blocco funzione non supportato.
- 9942050 Errore allocazione blocco funzione.
- 9942060 Terminato spazio memoria rilocabile, non è possibile eseguire l'"FB".
- 9942070 Errore versione blocco funzione.
- 9942990 Non implementata nel simulatore.

Esempi

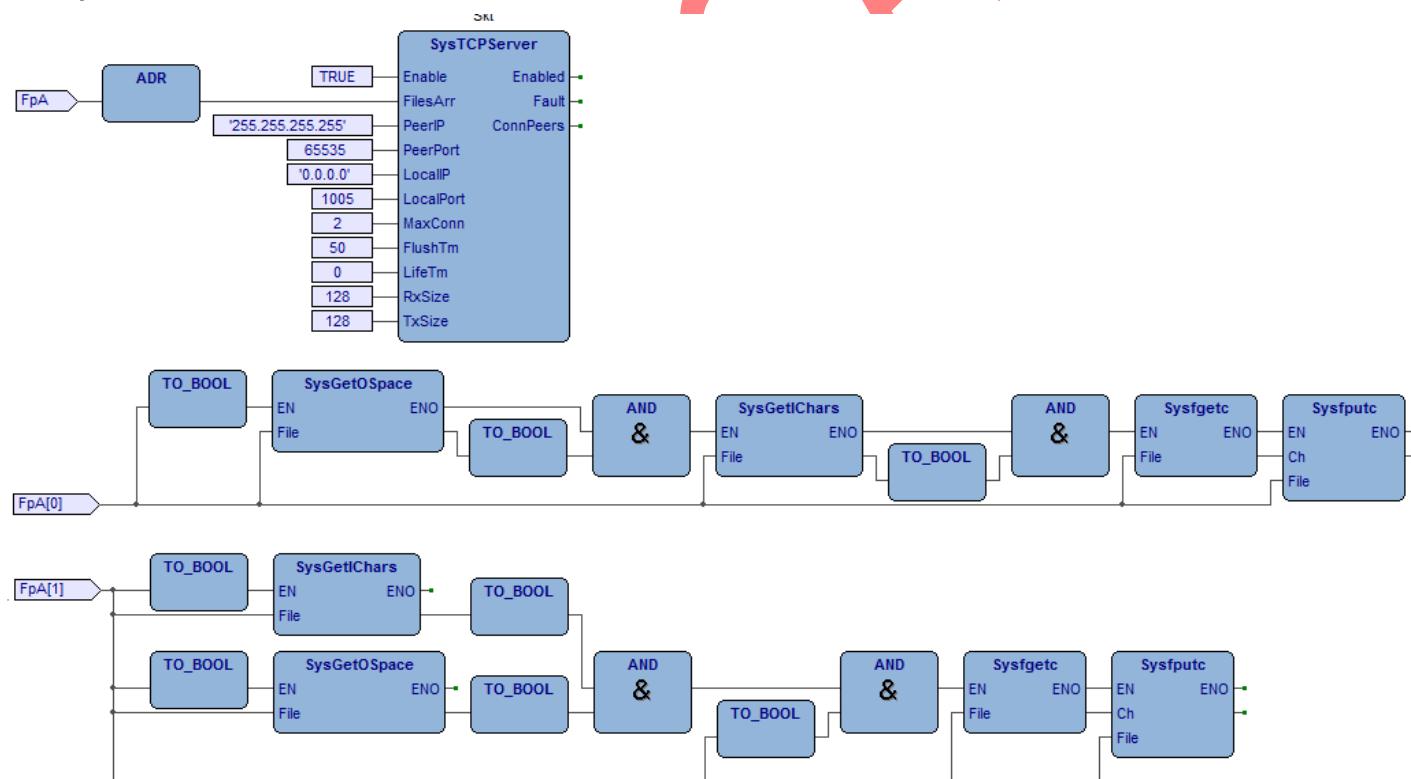
Nell'esempio è attivato un server TCP in ascolto sulla porta 1005. Il server accetta massimo 2 connessioni, su ogni connessione viene eseguito l'echo del carattere ricevuto. Connnettendosi in telnet alla porta 1005 inviando un carattere se ne riceve l'echo.

Come si vede dall'esempio la gestione dell'echo è stata realizzata in due modi diversi per i due sockets in ascolto.

Definizione variabili

	Name	Type	Address	Array	Init value	Attribute	Description
1	Skt	SysTCPServer	Auto	No	TCP socket server
2	FpA	FILEP	Auto	[0..1]	File pointer array

Esempio FBD (PTP119B000, FBD_TCPServerEcho)



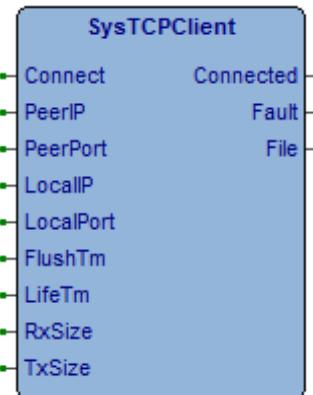
7.17.2 SysTCPClient, opens a TCP/IP connection

Type	Library
FB	XTarget_12_0_0

Questa function block gestisce la comunicazione con protocollo TCP/IP in modalità client.

Occorre definire l'indirizzo IP **PeerIP** e la porta TCP **PeerPort** del sistema server a cui ci si vuole connettere. Attivando il comando **Connect** viene aperta la connessione con il sistema server. Se la connessione va a buon fine viene attivato **Connected** e sull'uscita **File** viene ritornato lo stream da utilizzarsi per lo scambio dati con il sistema server.

Se la connessione non è possibile viene generato **Fault**.



Connect (BOOL)	Comando abilitazione connessione
PeerIP (STRING[15])	Indirizzo IP del sistema server a cui connettersi.
PeerPort (UINT)	Numero porta TCP a cui connettersi.
LocalIP (STRING[15])	Indirizzo IP della interfaccia di rete da cui effettuare la connessione. Default '0.0.0.0' : l'interfaccia è scelta automaticamente in base all'IP a cui connettersi.
LocalPort (UINT)	Numero porta TCP da cui parte la connessione (0 scelta automaticamente).
FlushTm (UINT)	Tempo di flush dati, se non sono caricati dati sullo stream dopo il tempo definito i dati presenti vengono automaticamente inviati (ms).
LifeTm (UINT)	Tempo di vita socket, se non sono ricevuti o inviati dati dopo il tempo definito il socket viene automaticamente chiuso (Sec).
RxSize (UINT)	Dimensione buffer ricezione dati.
TxSize (UINT)	Dimensione buffer trasmissione dati.
Connected (BOOL)	Attivo se connessione stabilita con server.
Fault (BOOL)	Attivo se errore gestione.
File (FILEP)	Stream di I/O. Viene valorizzato su connessione stabilita con il sistema server.

Codici di errore

In caso di errore si attiva l'uscita **Fault**, con [**SysGetLastError**](#) è possibile rilevare il codice di errore.

Codice Descrizione

- 9974005 Blocco funzione non supportato.
- 9974050 Errore allocazione blocco funzione.
- 9974060 Terminato spazio memoria rilocabile, non è possibile eseguire l'FB.
- 9974070 Errore versione blocco funzione.
- 9974990 Non implementata nel simulatore.

Esempi

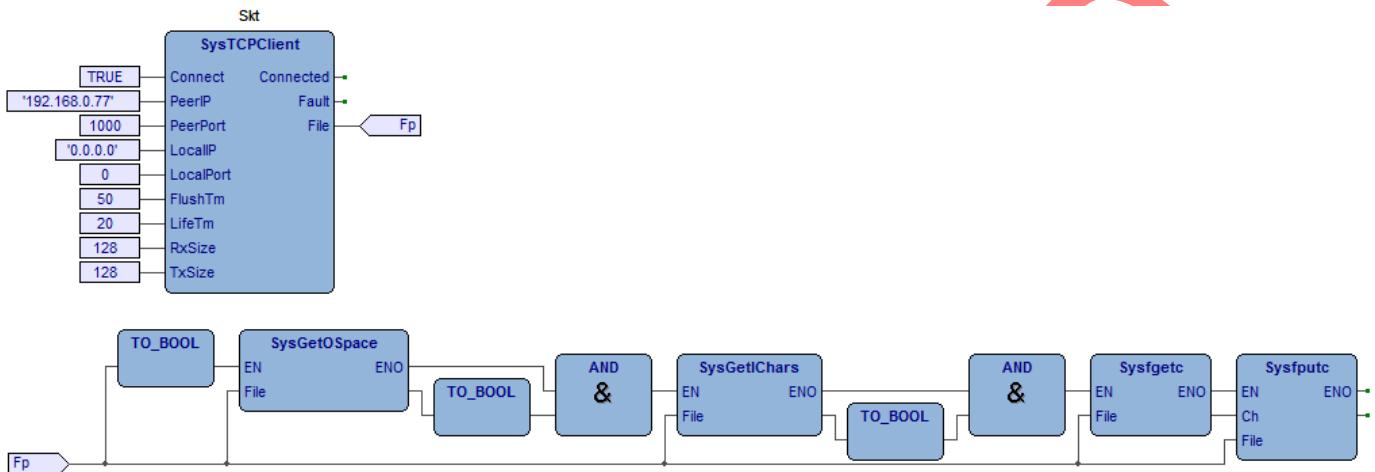
Nell'esempio è attivata una connessione verso un server TCP in ascolto sulla porta 1000. Eseguita la connessione i caratteri ricevuti dal server sono reinviati in echo.

Come server di prova è possibile utilizzare il programma Toolly scaricabile dal nostro sito.

Definizione variabili

	Name	Type	Address	Array	Init value	Attribute	Description
1	Skt	SysTCPClient	Auto	No	..		TCP socket client
2	Fp	FILEP	Auto	No	..		File pointer array

Esempio FBD (PTP119B000, FBD_TCPClientEcho)



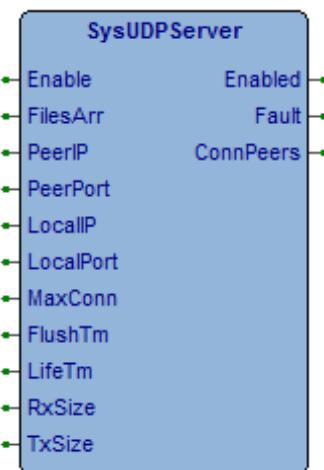
7.17.3 SysUDPServer, accepts UDP connections

Type	Library
FB	XTarget_12_0

Questa function block gestisce la comunicazione con protocollo UDP in modalità server. Occorre fornire l'indirizzo di un array di flussi dati streams al parametro **FilesArr**. Occorre definire la porta UDP da porre in ascolto ed il numero di connessioni contemporanee accettate.

Attivando il comando **Enable** il server UDP viene posto in ascolto sulla porta indicata, sulla connessione di un client viene incrementato il numero di **ConnPeers** ed uno degli stream definiti in **FileArr** viene valorizzato ed aperto. Sullo stream aperto è possibile utilizzare le funzioni di TermIO per gestire la comunicazione.

Per modificare i parametri occorre disattivare e poi riabilitare il comando **Enable**.



Enable (BOOL)	Comando abilitazione server
FilesArr (@FILEP)	Pointer ad array streams di I/O. I vari file streams saranno valorizzati alla connessione dei clients. Occorre definire un numero di streams pari al numero di connessioni contemporanee accettate.
PeerIP (STRING[15])	Range indirizzi IP da cui è accettata la connessione. La connessione è accettata se indirizzo IP del peer in AND con il valore non viene modificato. Default 255.255.255.255 : connessione accettata da tutti gli indirizzi IP.
PeerPort (UINT)	Range porte da cui è accettata la connessione. La connessione è accettata se porta del peer in AND con il valore non viene modificata. Default 65535 : connessione accettata da tutte le porte.
LocalIP (STRING[15])	Indirizzo IP della interfaccia di rete da cui si accettano le connessioni. Default '0.0.0.0' : le connessioni sono accettate da tutte le interfacce.
LocalPort (UINT)	Numero di porta in ascolto sul server.
MaxConn (USINT)	Numero massimo di connessioni contemporanee accettate dal server. Deve essere uguale al numero di files definiti.
FlushTm (UINT)	Tempo di flush dati, se non sono caricati dati sullo stream dopo il tempo definito i dati presenti vengono automaticamente inviati (ms).
LifeTm (UINT)	Tempo di vita socket, se non sono ricevuti o inviati dati dopo il tempo definito il socket viene automaticamente chiuso (Sec). Se definito tempo "0" il socket non viene mai chiuso.
RxSize (UINT)	Dimensione buffer ricezione dati.
TxSize (UINT)	Dimensione buffer trasmissione dati.
Enabled (BOOL)	Attivo se TCP server correttamente impostato e pronto.
Fault (BOOL)	Attivo se errore gestione.
ConnPeers (FILEP)	Numero di clients connessi al server.

Codici di errore

In caso di errore si attiva l'uscita **Fault**, con [**SysGetLastError**](#) è possibile rilevare il codice di errore.

Codice Descrizione

- 9945005 Blocco funzione non supportato.
- 9945050 Errore allocazione blocco funzione.
- 9945070 Errore versione blocco funzione.
- 9945990 Non implementata nel simulatore.

Esempi

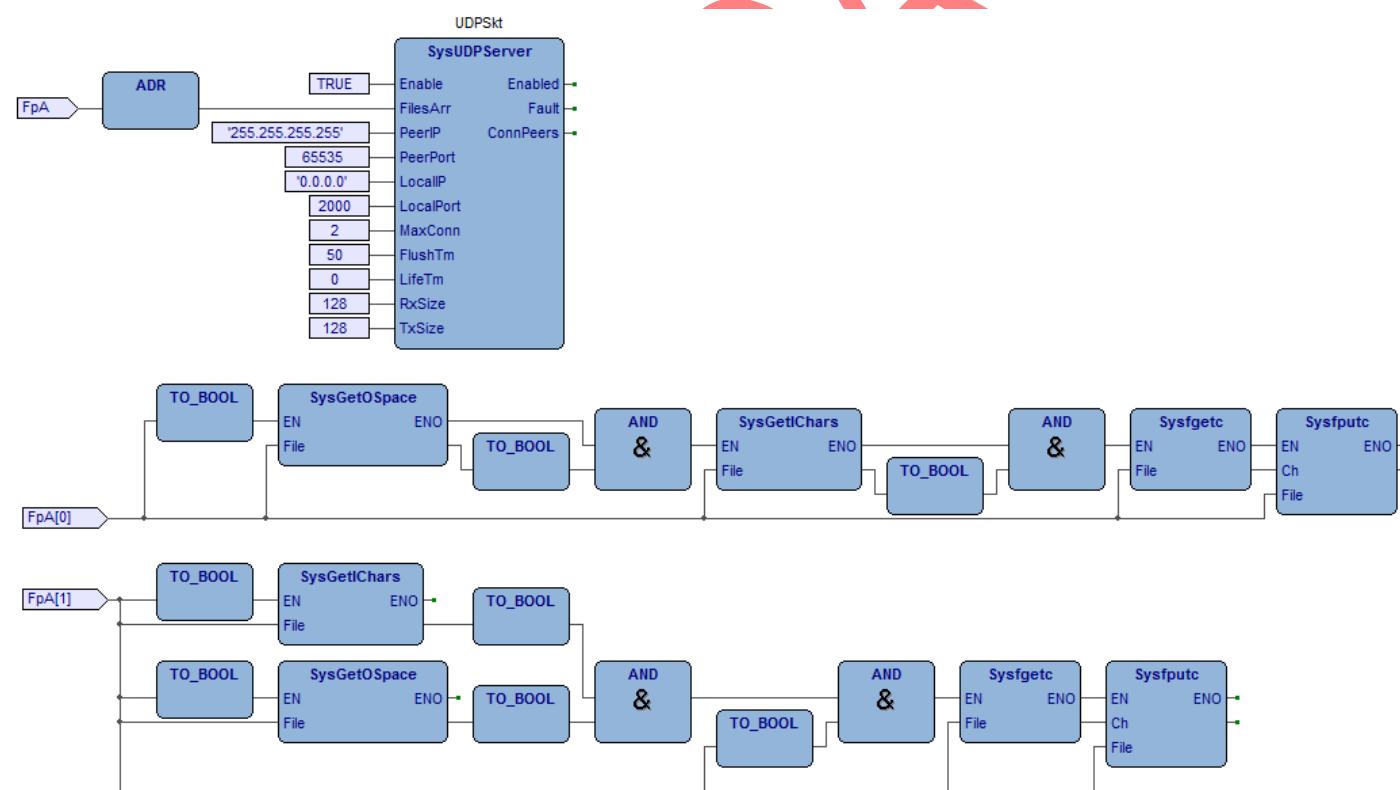
Nell'esempio è attivato un server UDP in ascolto sulla porta 2000. Il server accetta massimo 2 connessioni, su ogni connessione viene eseguito l'echo del carattere ricevuto. Connettendosi in telnet alla porta 1005 inviando un carattere se ne riceve l'echo.

Come si vede dall'esempio la gestione dell'echo è stata realizzata in due modi diversi per i due sockets in ascolto.

Definizione variabili

	Name	Type	Address	Array	Init value	Attribute	Description
1	UDPSkt	SysUDPServe	Auto	No	UDP socket server
2	FpA	FILEP	Auto	[0..1]	File pointer array

Esempio FBD (PTP119B000, FBD_UDPServerEcho)



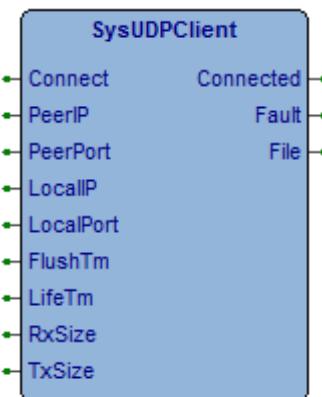
7.17.4 SysUDPCClient, opens a UDP connection

Type	Library
FB	XTarget_12_0

Questa function block gestisce la comunicazione con protocollo UDP in modalità client.

Occorre definire l'indirizzo IP **PeerIP** e la porta TCP **PeerPort** del sistema server a cui ci si vuole connettere. Attivando il comando **Connect** viene aperta la connessione con il sistema server. Se la connessione va a buon fine viene attivato **Connected** e sull'uscita **File** viene ritornato lo stream da utilizzarsi per lo scambio dati con il sistema server.

Se la connessione non è possibile viene generato **Fault**.



Connect (BOOL)	Comando abilitazione connessione
PeerIP (STRING[15])	Indirizzo IP del sistema server a cui connettersi.
PeerPort (UINT)	Numero porta TCP a cui connettersi.
LocalIP (STRING[15])	Indirizzo IP della interfaccia di rete da cui effettuare la connessione. Default '0.0.0.0' : l'interfaccia è scelta automaticamente in base all'IP a cui connettersi.
LocalPort (UINT)	Numero porta TCP da cui parte la connessione (65535 scelta automaticamente).
FlushTm (UINT)	Tempo di flush dati, se non sono caricati dati sullo stream dopo il tempo definito i dati presenti vengono automaticamente inviati (ms).
LifeTm (UINT)	Tempo di vita socket, se non sono ricevuti o inviati dati dopo il tempo definito il socket viene automaticamente chiuso (Sec).
RxSize (UINT)	Dimensione buffer ricezione dati.
TxSize (UINT)	Dimensione buffer trasmissione dati.
Connected (BOOL)	Attivo se connessione stabilita con server.
Fault (BOOL)	Attivo se errore gestione.
File (FILEP)	Stream di I/O. Viene valorizzato su connessione stabilita con il sistema server.

Codici di errore

In caso di errore si attiva l'uscita **Fault**, con [**SysGetLastError**](#) è possibile rilevare il codice di errore.

Codice	Descrizione
9943005	Blocco funzione non supportato.
9943050	Errore allocazione blocco funzione.
9943060	Terminato spazio memoria rilocabile, non è possibile eseguire l'FB.
9943070	Errore versione blocco funzione.
9943990	Non implementata nel simulatore.

Esempi

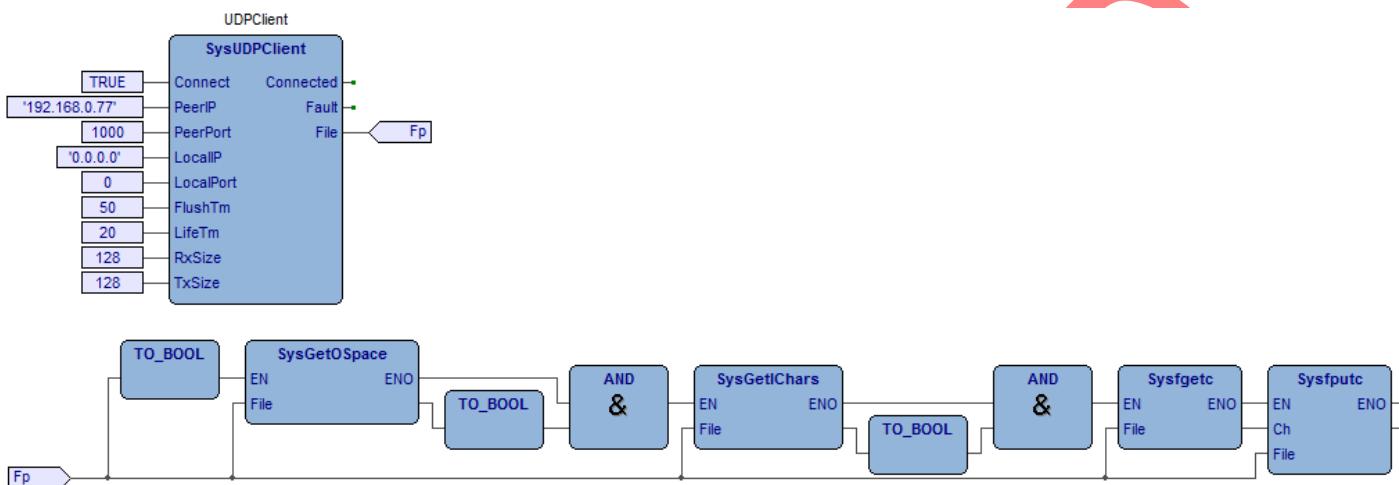
Nell'esempio è attivata una connessione verso un server TCP in ascolto sulla porta 1000. Eseguita la connessione i caratteri ricevuti dal server sono reinviati in echo.

Come server di prova è possibile utilizzare il programma Toolly scaricabile dal nostro sito.

Definizione variabili

	Name	Type	Address	Array	Init value	Attribute	Description
1	UDPClient	SysUDPCClient	Auto	No	..		UDP socket client
2	Fp	FILEP	Auto	No	..		File pointer array

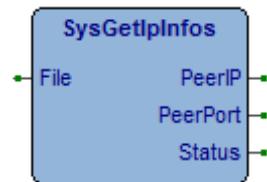
Esempio FBD (PTP119B000, FDB_UDPClientEcho)



7.17.5 SysGetIpInfos, returns IP infos

Type	Library
FB	XTarget_12_0

Questo blocco funzione ritorna le informazioni della connessione. Passando al blocco funzione un File di tipo TCP o UDP è possibile avere in uscite le informazioni relative.



File (FILEP) File pointer (Deve essere di tipo TCP o UDP).

PeerIP (STRING[15]) Stringa di definizione indirizzo IP del peer connesso al file.

PeerPort (UINT) Porta del peer connesso al file.

Status (DWORD) Stato connessione (Non gestito)

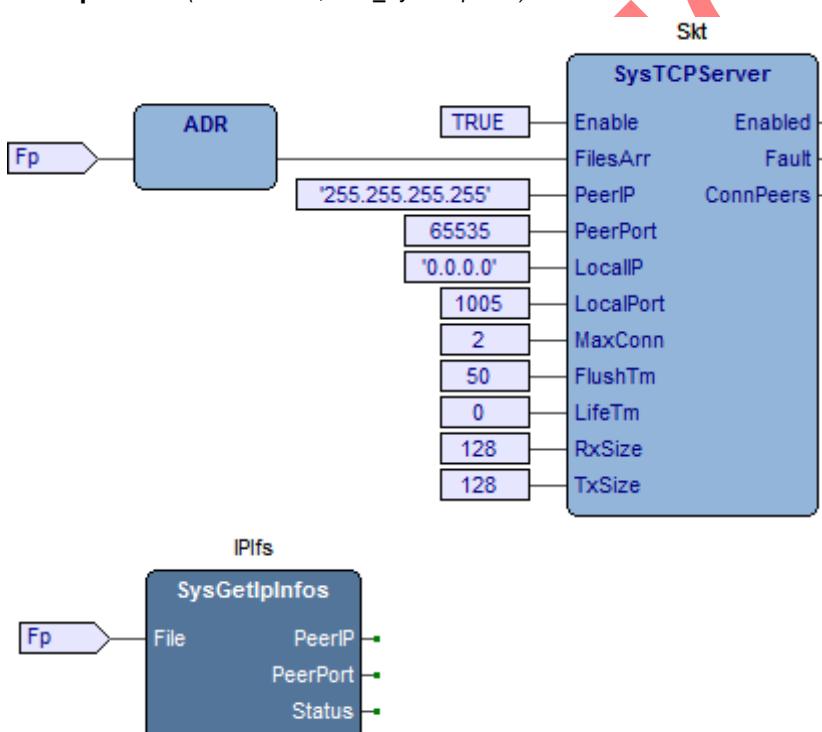
Esempi

Nell'esempio sono visualizzate le informazioni di connessione al socket.

Definizione variabili

	Name	Type	Address	Array	Init value	Attribute	Description
1	Skt	SysTCPServer	Auto	No	TCP socket server
2	IPIfs	SysGetIpInfos	Auto	No	Get IP infos
3	Fp	FILEP	Auto	No	File pointer

Esempio FBD (PTP119B000, FBD_SysGetIpInfos)



7.17.6 SysIPReach, IP address is reachable

Type	Library
FB	XTarget_07_0

This function block performs the check if an IP address is reachable. A Ping command is sent to the system and if it's been answered **Done** is activated.

The **Refresh** variable returns the percentage of time elapsed since the last execution of the Ping command. Reached the 50% of the time (Approximately 25 seconds) a new Ping command is sent.



Enable (BOOL)	Enables function block. Activating it, a Ping command will be send every 25 Sec.
PeerIP (STRING[15])	IP address to reach.
Done (BOOL)	Activated if IP address is reachable (Ping command is been answered).
Fault (BOOL)	Active for a program loop if there is an error.
Refresh (USINT)	Percentage of time from last Ping command.

Error codes

If an error occurs, the **Fault** output is activated and [SysGetLastError](#) can detect the error code.

- 9974005 Function block not supported.
- 9974050 Function block allocation error.
- 9974060 Relocatable memory space full. You can not run the FB.
- 9974070 Function block version error.
- 9974100 FB executed not in the Back task.
- 9974110 Error on defined **PeerIP** address.
- 9974200 Error sending Ping request.
- 9974300 Error in Ping answer.
- 9974990 Not yet Implemented in the simulator.

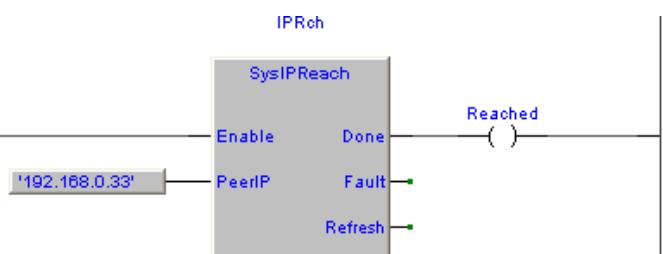
Examples

The example checks whether the IP address **192.168.0.33** is reachable. In this case the **Reached** output is activated.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Reached	BOOL	Auto	No	FALSE	..	IP address reached
2	IPRch	SysIPReach	Auto	No	0	..	FB IP reach

LD example



7.17.7 UDPDataTxfer, UDP data transfer

Type	Library
FB	eLLabNetworkLib_A000

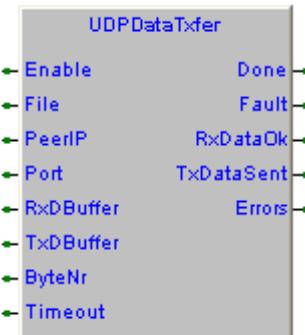
This function block performs the transfer of a block of memory between two systems using UDP over Ethernet connection. It is necessary to pass to **File** variable a data stream, previously opened by the [Sysfopen](#) function, and the socket must have been set in listening status using the [SysSktListen](#) function.

The **PeerIP** parameter indicates the IP address to which the data will be sent. The **Port** parameter indicates the port used to transfer data (It must be the same for both systems).

The **Timeout** parameter defines the maximum time for data transfer. Delivery of data ends with the receipt of an acknowledge from the other system. A cycle of sending data and receiving acknowledge requires 2 loops of program. If after sending data, the Ack is not received within a timeout period equal to **Timeout/4**, a new try is started and so on until the end of the defined time. To guarantee at least 3 retries **it is recommended to set as a Timeout value of 10 times the maximum loop** (you choose the higher from the two communication systems).

The data is sent automatically when a change in any of the bytes of the buffer is detected. To ensure control over the connection between the two systems, each **Timeout** time the buffer data is sent.

If the two systems are communicating, the **Done** output is activated. **RxDataOk** will be activated for a program loop every time you receive data from the other system, while **TxDatasent** is activated for a program loop at the end of the transmission of data buffer to the other system.



Enable (BOOL)	Enables function block.
File (FILEP)	Stream returned by Sysfopen function.
PeerIP (STRING[15])	IP address of server to which to send data.
Port (UINT)	Port used to transfer data (It must be the same for both systems).
RxDBuffer (@USINT)	Pointer to buffer where to transfer the received data.
TxDBuffer (@USINT)	Pointer to buffer containing data to send.
ByteNr (UDINT)	Number of exchanged bytes.
Timeout (UINT)	Maximum time to transfer data (mS).
Done (BOOL)	Activated when both systems are in communication.
Fault (BOOL)	Active for a program loop if there is an error.
RxDataOk (BOOL)	Active for a program loop when data are received from other system.
TxDatasent (BOOL)	Active for a program loop at the end of data transmission to other system.
Errors (UINT)	Number of errors. Incremented every new error occurs. When it reaches the maximum value, it restart from 0.

Error codes

If an error occurs, the **Fault** output is activated and [SysGetLastError](#) can detect the error code.

- 10014050 **File** value not defined.
- 10014100 Relocatable memory space full. You can not run the FB.
- 10014200~2 Error receiving data frame.
- 10014300~2 Error receiving acknowledge.
- 10014400 Received an unknown command.
- 10014500~1 Error on transmission sequences.
- 10014600 Timeout on sending data.

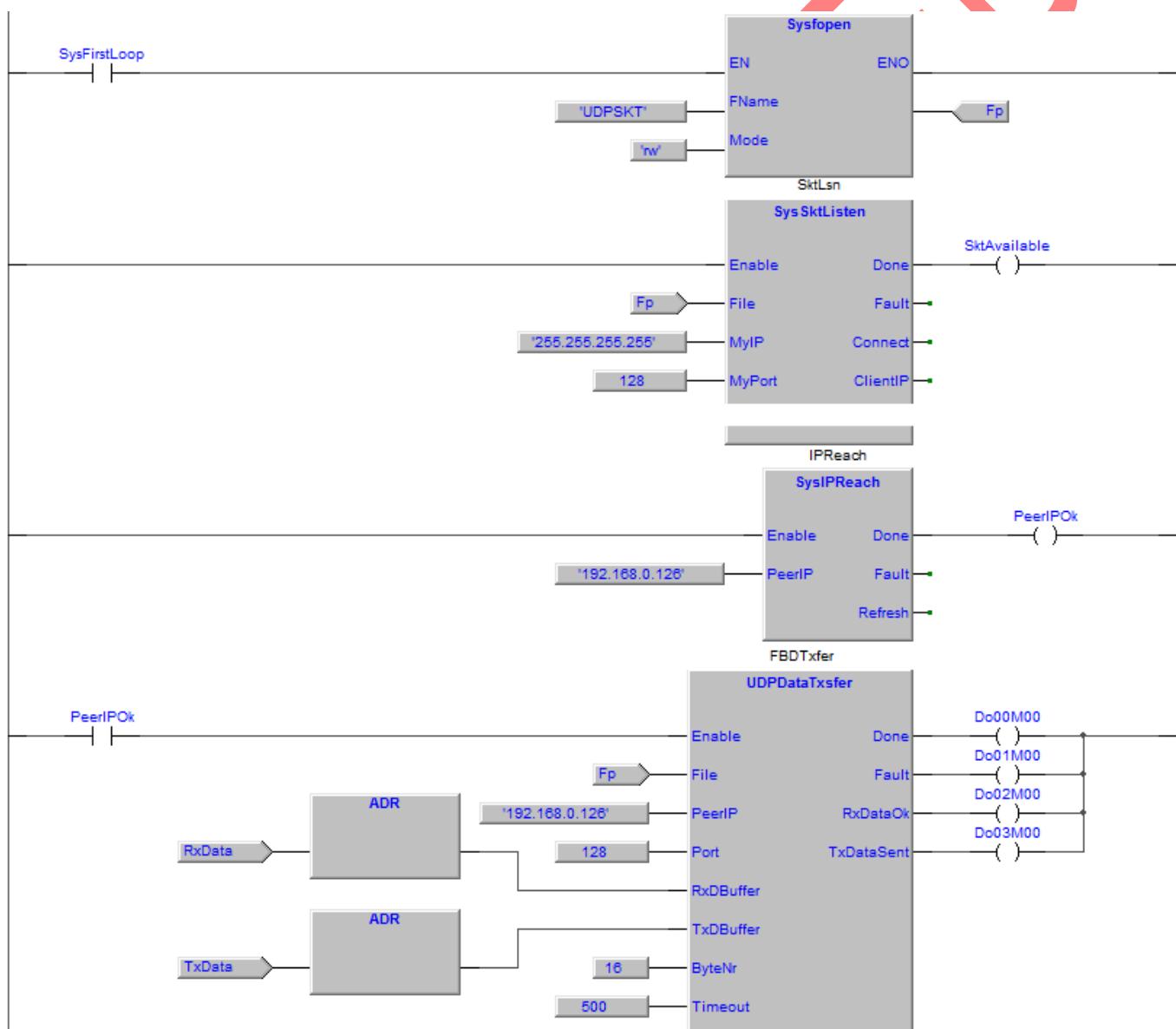
Examples

In the example, a block of 16 bytes of memory is exchanged to the system with IP 192.168.0.126.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	PeerIPok	BOOL	Auto	No	FALSE	..	Peer IP is reached
2	RxData	BOOL	Auto	[0..15]	16(0)	..	Rx data buffer
3	SktAvailable	BOOL	Auto	No	FALSE	..	Socket available
4	TxDATA	BOOL	Auto	[0..15]	16(0)	..	Tx data buffer
5	Fp	FILEP	Auto	No	0	..	UDP socket
6	IPReach	SysIPReach	Auto	No	0	..	IP reach (Ping)
7	SktLsn	SysSktListen	Auto	No	0	..	Socket listen
8	FBDTxfer	UDPDATATXFER	Auto	No	0	..	UDP data transfer

LD example (PTP119A300, UDPDataTransfer)



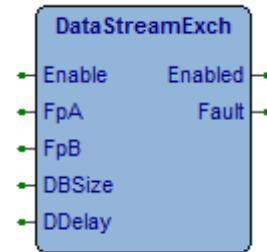
7.17.8 DataStreamExch, exchanges data between two I/O streams

Type	Library
Function	eLLabNetworkLib_A000

This function block performs data exchange between two I/O streams. By defining a stream as a TCP socket and the other as serial port, it is possible to realize a Ethernet/Serial converter.

In **FpA** and **FpB** must indicate the file pointer in output from the **Sysfopen** function opening the I/O stream. The FB receives data from a stream and stores them in a buffers of **DBSize** size, when non data is received for a **DDelay** sends them to the other stream.

If are received more data than the defined size of the buffer, the data is immediately sent on the other stream and an error is generated.



Enable (BOOL) Enables the FB.

FpA (FILEP) *Stream* returned from **Sysfopen** function.

FpB (FILEP) *Stream* returned from **Sysfopen** function.

DBSize (UDINT) Data buffer size (Bytes).

DDelay (UDINT) Receiving characters, pause time (mS).

Enabled (BOOL) FB enabled.

Fault (BOOL) Active for a loop on error.

Error codes

If an error occurs, the **Fault** output is activated and [**SysGetLastError**](#) can detect the error code.

10048100 Data buffer allocation error.

10048200 Data buffer full on data reception from stream (A).

10048250 Data buffer full on data reception from stream (B).

Examples

A simple Ethernet/Serial converter, accepts TCP connection on port 1000 and uses the COM0 serial port.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Fp	FILEP	Auto	No	..		File pointer
2	Sp	SysSerialPort	Auto	No	..		Serial port management
3	Sk	SysTCPServer	Auto	No	..		Socket server
4	DExch	DataStreamExch	Auto	No	..		Data stream exchange

ST example (PTP114A640, ST_DataStreamExch)

```
(* Eseguo inizializzazioni. *)
IF (SysFirstLoop) THEN
    (* Inizializzo porta seriale. *)
    Sp.COM:='COM2'; (* Porta seriale *)
    Sp.Baudrate:=19200; (* Baud rate *)
    Sp.Parity:='E'; (* Parity *)
    Sp.DataBits:=8; (* Data bits *)
    Sp.StopBits:=1; (* Bit di stop *)
    Sp.DTRManagement:=DTR_AUTO_WO_TIMES; (* DTR management *)
    (* Inizializzo socket server. *)
    Sk.FilesArr:=ADR(Fp); (* File array *)
    Sk.MaxConn:=1; (* Numero connessioni accettate *)
    Sk.LocalIP:='0.0.0.0'; (* Local IP *)
    Sk.LocalPort:=2000; (* Local port *)
    Sk.PeerIP:='255.255.255.255'; (* Accetto tutti gli IP *)
    Sk.PeerPort:=65535; (* Accetto tutte le porte *)
    Sk.LifeTm:=60; (* Tempo di vita socket (S) *)
    Sk.FlushTm:=10; (* Tempo di flush socket (mS) *)
    Sk.RxSize:=512; (* Rx size buffer *)
    Sk.TxSize:=512; (* Tx size buffer *)
    (* Inizializzo scambio dati tra streams. *)
    DExch.DBSize:=256; (* Data buffer size *)
    DExch.DDelay:=500; (* Data delay (mS) *)
END_IF;
(* ----- *)
(* GESTIONE SCAMBIO DATI DA TCP E RTU *)
(* ----- *)
(* Gestione scambio dati tra Modbus TCP e Modbus RTU. *)

Sp(Open:=TRUE); (* Gestione porta seriale *)
Sk(Enable:=TRUE); (* Gestione TCOP server *)
DExch.FpA:=Sp.File; (* File pointer *)
DExch.FpB:=Fp; (* File pointer *)
DExch(Enable:=(Sk.ConnPeers <> 0)); (* Modbus TCP gateway *)

(* [End of file] *)
```

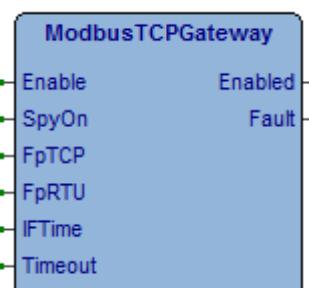
7.17.9 ModbusTCPGateway, modbus TCP gateway

Type	Library
FB	eLLabNetworkLib_A000

This function block operates as Modbus gateway between Modbus TCP connection and a serial Modbus RTU. In **FpTCP** must pass the I/O stream from which arrive the Modbus TCP requests. In **FpRTU** must pass the I/O stream to which are sent the converted to Modbus RTU requests.

In **Iftime** must be defined the pause time of receiving characters from the serial port (Modbus RTU). **Timeout** defines the maximum execution time of a command Modbus TCP (From when the command is received to when after conversion the Modbus RTU response is returned).

The **SpyOn** if active allows to spy the FB operation. In case of execution error or execution time greater than the time set in Timeout, is activated for a program loop the **Fault** output.



Enable (BOOL) Command that enables the FB.

SpyOn (BOOL) Active allows to spy the FB working.

FpTCP (FILEP) Stream returned from **Sysopen** function.

FpRTU (FILEP) Stream returned from **Sysopen** function.

IFTIME (UDINT) Time between frame (μ S), it must be related to baud rate.

Baud rate	Tempo
300	112000
600	56000
1200	28000
2400	14000
4800	7000
9600	3430

Baud rate	Tempo
19200	1720
38400	860
57600	573
76800	429
115200	286

Timeout (UINT) Maximum command execution time expressed in mS. If the command does not end in the defined time, the command is aborted and the **Fault** output is activated

Enabled (BOOL) FB enabled.

Fault (BOOL) Active for a program loop if execution error.

Spy trigger

If **SpyOn** is active the **SysSpyData** function is executed this allows to spy the FB operations. There are various levels of triggers.

TFlags

Descrizione

16#00000001 **Tx:** Modbus RTU command frame sent.

16#00000002 **Rx:** Modbus RTU answer frame received.

Error codes

If an error occurs, the **Fault** output is activated and **SysGetLastError** can detect the error code..

10053050 Execution timeout

10053100~1 Modbus TCP frame management error.

10053200~1 Modbus RTU frame management error.

Examples

Simple Modbus TCP/RTU gateway, accepts TCP connection on port 2000 and uses the serial port COM2.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Fp	FILEP	Auto	No	..		File pointer
2	Sp	SysSerialPort	Auto	No	..		Serial port management
3	Sk	SysTCPServer	Auto	No	..		Socket server
4	DExch	ModbusTCPGateway	Auto	No	..		Modbus TCP gateway

ST example (PTP114A640, ST_ModbusTCPGateway)

```

IF (SysFirstLoop) THEN
    (* Inizializzo porta seriale. *)
    Sp.COM:='COM2'; (* Porta seriale *)
    Sp.Baudrate:=19200; (* Baud rate *)
    Sp.Parity:='E'; (* Parity *)
    Sp.DataBits:=8; (* Data bits *)
    Sp.StopBits:=1; (* Bit di stop *)
    Sp.DTRManagement:=DTR_AUTO_WO_TIMES; (* DTR management *)

    (* Inizializzo socket server. *)
    Sk.FilesArr:=ADR(Fp); (* File array *)
    Sk.MaxConn:=1; (* Numero connessioni accettate *)
    Sk.LocalIP:='0.0.0.0'; (* Local IP *)
    Sk.LocalPort:=2000; (* Local port *)
    Sk.PeerIP:='255.255.255.255'; (* Accetto tutti gli IP *)
    Sk.PeerPort:=65535; (* Accetto tutte le porte *)
    Sk.LifeTm:=60; (* Tempo di vita socket (S) *)
    Sk.FlushTm:=10; (* Tempo di flush socket (mS) *)
    Sk.RxSize:=512; (* Rx size buffer *)
    Sk.TxSize:=512; (* Tx size buffer *)

    (* Inizializzo scambio dati tra Modbus TCP e Modbus RTU. *)
    DExch.IFTime:=1720; (* Tempo ricezione caratteri *)
    DExch.Timeout:=500; (* Timeout esecuzione comando (mS) *)
END_IF;

(* ----- *)
(* GESTIONE SCAMBIO DATI DA TCP E RTU *)
(* ----- *)
(* ----- *)
(* Gestione scambio dati tra Modbus TCP e Modbus RTU. *)

Sp(Open:=TRUE); (* Gestione porta seriale *)
Sk(Enable:=TRUE); (* Gestione TCOP server *)
DExch.FpRTU:=Sp.File; (* File pointer (Modbus RTU) *)
DExch.FpTCP:=Fp; (* File pointer (Modbus TCP) *)
DExch(Enable:=(Sk.ConnPeers >> 0)); (* Modbus TCP gateway *)

(* [End of file] *)

```

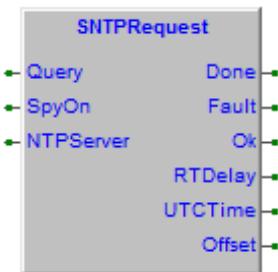
7.17.10 SNTPRequest, sends a SNTP request

Type	Library
FB	eLLabNetworkLib_A000

This function block reads from a time server the UTC time. By activating the **Query** command a request is sent to the server passed in **NTPServer**.

The FB queries the server and if answered correctly **UTCTime** returns the value of time in UTC Epoch. In **Offset** is returned in the difference between the UTC value returned and the NTP clock reference.

At the end of execution the **Done** output is activated, to make a new time reading must be deactivate and then reactivate the Query input. In **RTDelay** is returned the round trip delay, the time required to transmit the request and receive the response from the NTP server.



Query (BOOL)	Request activation command.
SpyOn (BOOL)	Active allows to spy the FB working.
NTPServer (STRING[15])	NTP server IP address definition.
Done (BOOL)	Active at the end of execution, is active even in case of Fault .
Ok (BOOL)	Active for a program loop if query correctly executed.
Fault (BOOL)	Active for a program loop if execution error.
RTDelay (REAL)	Round trip delay, time required to transmit the request and receive the response from the NTP server (mS).
UTCTime (UDINT)	Date/Time in UTC expressed in Epoch time.
Offset (REAL)	Difference between the UTC value returned and NTP reference clock (mS).

Spy trigger

If **SpyOn** is active the [SysSpyData](#) function is executed this allows to spy the FB operations. There are various levels of triggers.

TFlags Description

- 16#00000001 '--' NTP request informations.
- 16#00000002 '--' Reports of all operations performed.

Error codes

If an error occurs, the **Fault** output is activated and [SysGetLastError](#) can detect the error code.

- 10052010 FB not executed in the background task.
- 10052010 No more UDP sockets available.
- 10052100 Error on pinging NTP server.
- 10052110 Error socket in listening.
- 10052120 Error receiving data from the socket.
- 10014200~3 Error in the management sequences.

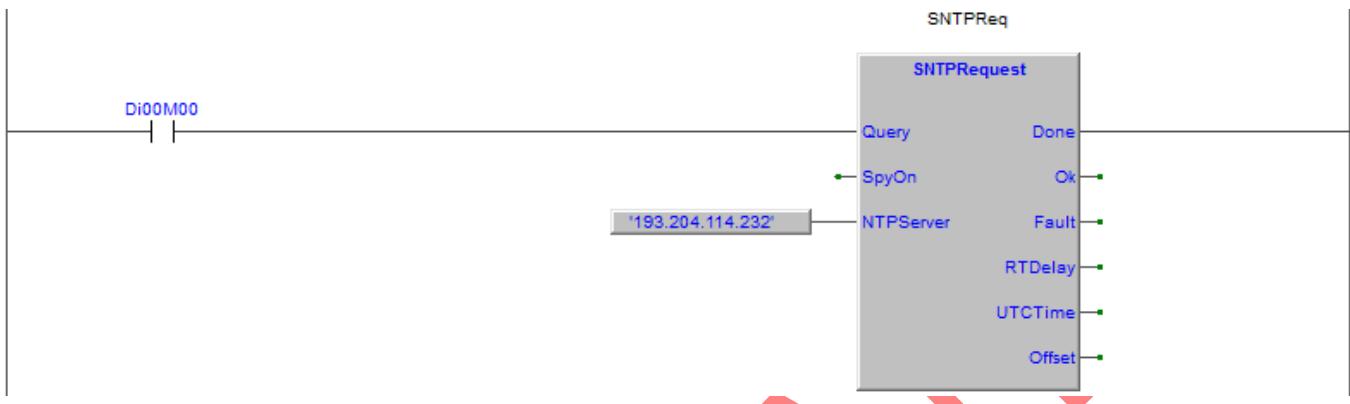
Examples

It's required the time from the ntp1.inrim.it (193 204 114 232) server.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	SNTPReq	SNTPRequest	Auto	No	..		FB SNTPRequest

LD example



DEPRECATION

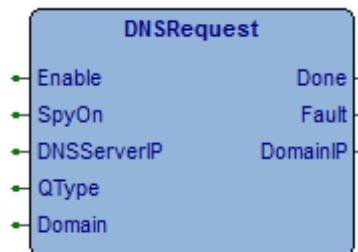
7.17.11 DNSRequest, sends a DNS request

Type	Library
FB	eLLabNetworkLib_A000

Questo blocco funzione invia una richiesta DNS. Indicando in **DNSServerIP** l'indirizzo IP di un server DNS (Esempio quello di Google 8.8.8.8) è possibile effettuare la richiesta di risoluzione di un dominio. Viene ritornato l'indirizzo IP ed il tempo di vita dell'URL.

Attivando l'ingresso **Enable** viene effettuata la richiesta di risoluzione dell'URL al server DNS. Se il server risolve l'URL viene attivata l'uscita **Done** e su **DomainIP** viene ritornato l'IP relativo.

Se **Enable** rimane attivo il FB controlla in tempo di vita della voce DNS ed esegue il riaggiornamento automatico dell'indirizzo IP. L'ingresso **SpyOn** se attivo permette di spiare il funzionamento del FB.



Enable (BOOL) Comando abilitazione esecuzione. Per rieseguire il comando disabilitare e poi riabilitare questo ingresso.

SpyOn (BOOL) Se attivo permette di spiare il funzionamento della FB.

DNSServerIP (STRING[15]) Stringa di definizione IP server DNS.

Domain (@USINT) Indirizzo stringa definizione URL di cui si desidera risolvere l'IP.

QType (UINT) Tipo di query.

Codice	Descrizione
16#0001	A record (host addresses).
16#0002	Name servers (NS) records (Not managed).
16#000F	Mail server (MX).

Done (BOOL) Si attiva al termine della esecuzione comando.

Fault (BOOL) Attivo per un loop se errore esecuzione comando.

Domain (STRING[15]) Stringa di definizione IP trovato.

RTime (UDINT) Tempo riferimento per controllo sul tempo di vita richiesta (uS).

TTL (UDINT) Tempo di vita richiesta (S).

Trigger di spy

Se **SpyOn** attivo viene eseguita la funzione **SysSpyData** che permette di spiare il funzionamento della FB. Sono previsti vari livelli di triggers.

TFlags Descrizione

16#00000001 **Tx:** Invio richiesta DNS.

16#00000002 **Rx:** Ricezione risposta DNS.

16#10000000 **Lg:** Messaggio di log.

16#20000000 **Wr:** Messaggio di warning.

16#40000000 **Er:** Messaggio di errore.

Codici di errore

In caso di errore si attiva l'uscita **Fault**, con [**SysGetLastError**](#) è possibile rilevare il codice di errore.

Codice Descrizione

- 10055010 FB eseguita in una task diversa dalla task di background.
- 10055020 **File** non corretto.
- 10055100 Errore lunghezza nome dominio da risolvere.
- 10055200 Timeout risposta da server.
- 10055210~5 Formato pacchetto ricevuto non corretto.
- 10055800 Errore esecuzione.

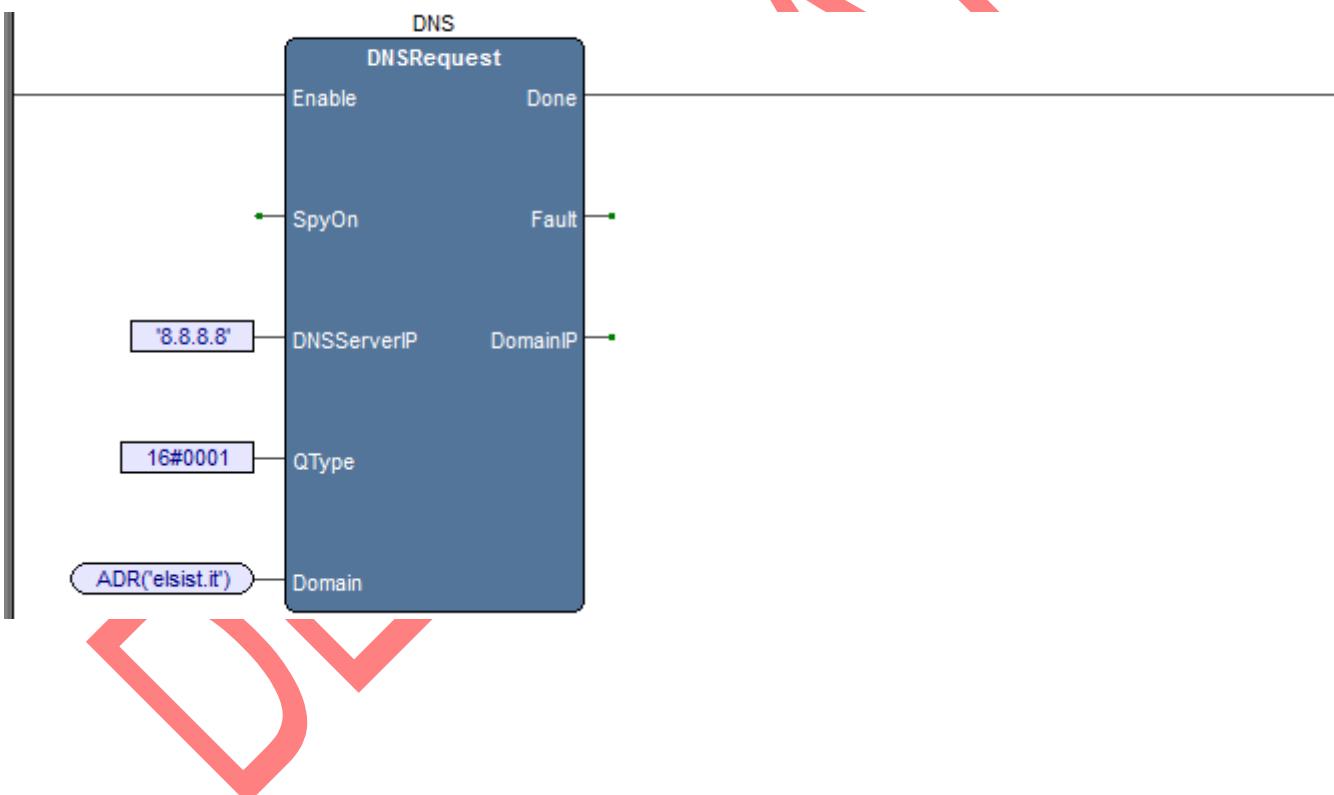
Esempi

Nell'esempio viene richiesto l'indirizzo IP di "elsist.it" al server DNS di Google.

Definizione variabili

	Name	Type	Address	Array	Init value	Attribute	Description
1	DNS	DNSRequest	Auto	No		..	

Esempio LD



7.17.12 HTTPProtocol, HTTP protocol management

Type	Library
FB	eLLabNetworkLib_A000

Questo blocco funzione esegue la richiesta di una pagina web con il protocollo HTTP. Su fronte attivazione ingresso **Enable** viene inviata la richiesta HTTP sullo stream di I/O definito in **File**. Ricevuta la pagina si attiva per un loop di programma l'uscita **Ok**.

Nel parametro **Host** occorre definire il nome host del server HTTP, mentre in **Page** occorre definire la pagina da richiedere. La pagina viene richiesta con i parametri definiti nel buffer **Request**. La pagina richiesta viene ritornata nel buffer definito in **Pbuffer**. In **PLength** è ritornata la lunghezza della pagina ricevuta.

In caso di errore esecuzione o tempo di esecuzione comando superiore al tempo definito in **Timeout**, viene attivata per un loop di programma l'uscita **Fault**.

L'uscita **Done** si attiva al termine della esecuzione della richiesta e su errore, per acquisire nuovamente la pagina occorre disabilitare e poi riabilitare l'ingresso **Enable**.



Enable (BOOL) Comando attivazione richiesta pagina.

SpyOn (BOOL) Se attivo permette di spiare il funzionamento della FB.

File (FILEP) Flusso dati stream ritornato dalla funzione **Sysopen**.

Host (STRING[128]) Stringa di definizione Host name.

Page (STRING[128]) Stringa di definizione pagina web richiesta.

Request (@USINT) Indirizzo buffer dati da inviare con la richiesta.

PBuffer (@USINT) Indirizzo buffer pagina ricevuta.

PBLength (UDINT) Dimensione buffer di pagina.

Timeout (UINT) Timeout esecuzione richiesta pagina (mS).

Done (BOOL) Attivo a fine esecuzione, si attiva anche in caso di **Fault**.

Ok (BOOL) Attivo per un loop di programma su ricezione pagina.

Fault (BOOL) Attivo per un loop di programma se errore gestione.

HttpStatus (STRING[64]) Status risposta HTTP ricevuta.

PLength (UINT) Dimensione pagina ricevuta.

PLTime (REAL) Tempo impiegato per caricamento pagina (S).

Codici di errore

In caso di errore si attiva l'uscita **Fault**, con [SysGetLastError](#) è possibile rilevare il codice di errore.

Codice Descrizione

10054010 File non definito

10054020 FB eseguita in una task diversa dalla task di background.

10054100 Timeout richiesta pagina.

10054200 Campo **Request** troppo lungo.

10054300~1 Errore ricezione lunghezza pagina.

10054300 Errore ricezione stringa risposta da server.

Esempi

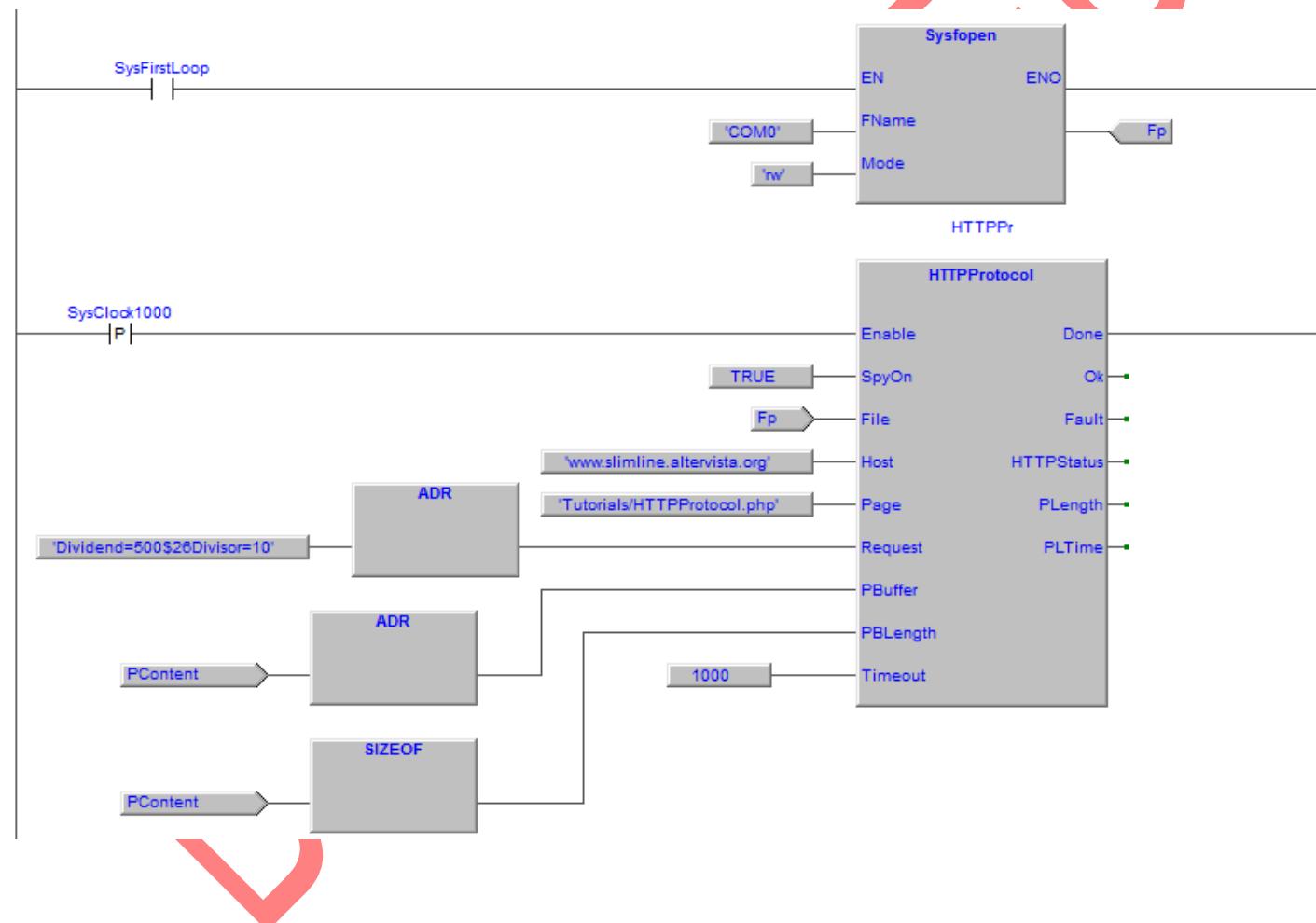
Nell'esempio viene eseguita la richiesta della pagina www.slimline.altervista.org/Tutorials/HTTPProtocol.php. Sono passati alla pagina 2 parametri **Dividend** e **Divisor**, la pagina è postata su Altervista ed al suo interno un semplice script PHP esegue la divisione tra i due valori passati e ritorna il risultato.

Se da un browser si indirizza la pagina www.slimline.altervista.org/Tutorials/HTTPProtocol.php?Dividend=500&Divisor=10 otterremo come ritorno la scritta "The result is: 50" che è esattamente quello che si trova in **PContent**.

Definizione variabili

	Name	Type	Address	Array	Init value	Attribute	Description
1	Fp	FILEP	Auto	No		..	File pointer
2	HTTPPr	HTTPProtocol	Auto	No		..	HTTPProtocol FB
3	PContent	STRING	Auto	[64]		..	Page content

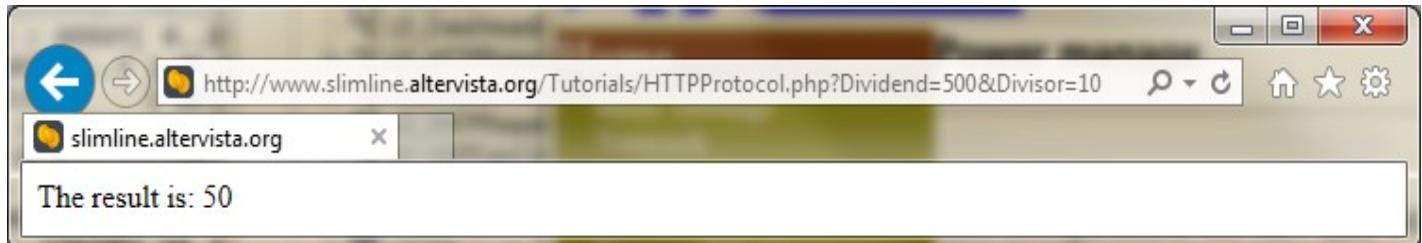
Esempio LD



Ecco il sorgente della pagina PHP caricata sul server Altervista..

```
<?php
echo "The result is: ".$_REQUEST["Dividend"] / $_REQUEST["Divisor"];
?>
```

Aprendo con un browser la pagina PHP si può testarne il funzionamento.



Ma per capire come l'esempio funziona occorre fare un preambolo. La versione attuale del sistema operativo dei sistemi SlimLine non permette di utilizzare i socket TCP in modalità client. Per potersi connettere al server di Altervista nell'esempio ho utilizzato un convertitore Ethernet/Seriale (Nel mio caso un ATC-3000) configurato come client TCP.

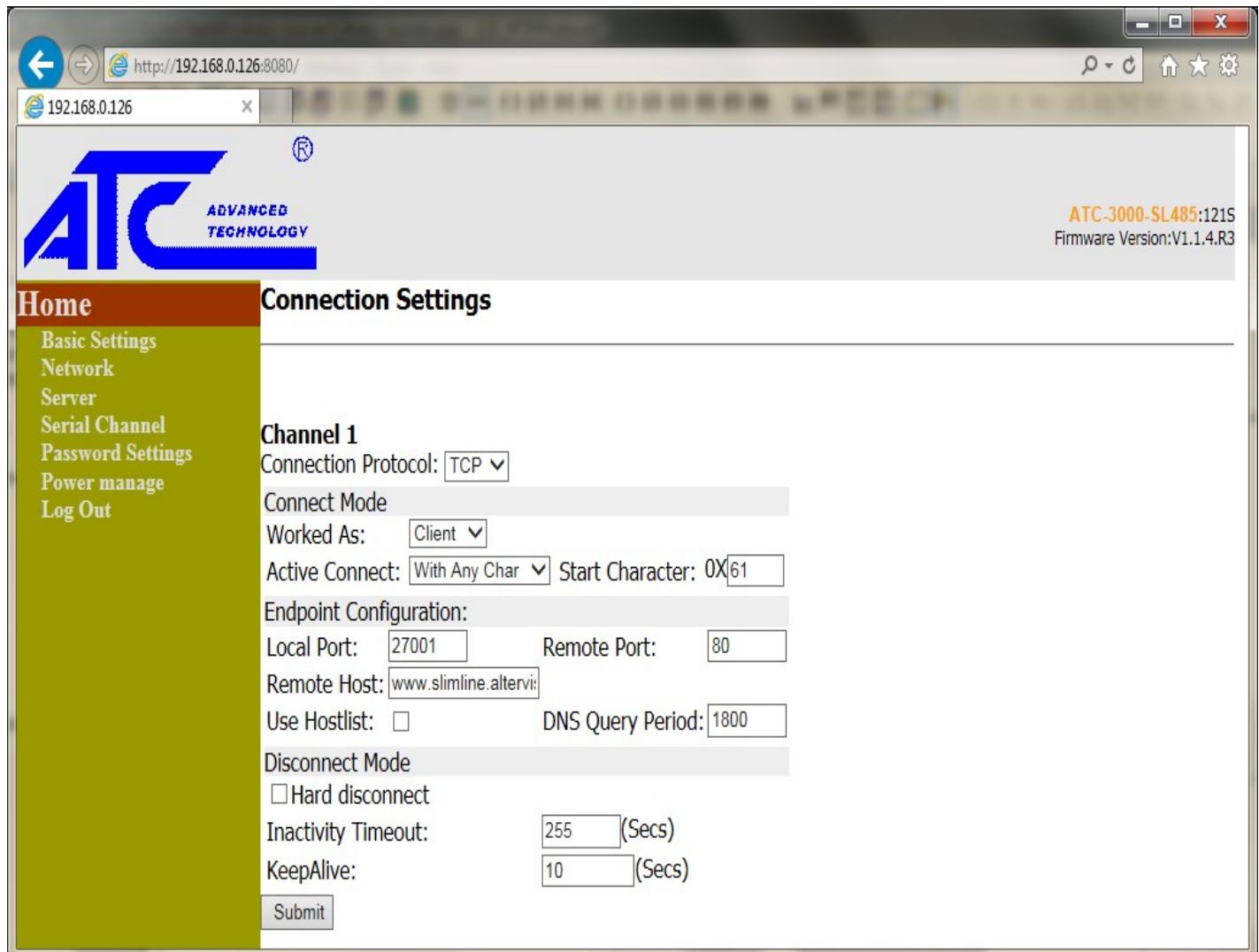
La porta seriale è settata con i parametri standard di default dei sistemi SlimLine **115200, e, 8, 1**.

The screenshot shows the configuration interface for an ATC-3000 SL485 device. The top bar shows the IP address <http://192.168.0.126:8080>. The main page displays the ATC logo and the text "ATC-3000-SL485:121S Firmware Version:V1.1.4.R3". On the left, a sidebar menu includes Home, Basic Settings, Network, Server, Serial Channel, Password Settings, Power manage, and Log Out. The main content area is titled "Serial Settings" and contains the following configuration:

- Channel 1**
- Enable Serial Port
- Port Settings**
- Protocol: RS232
- FIFO: 8
- Flow Control: None
- Baud Rate: 115200
- Data Bits: 8
- Parity: EVEN
- Stop bits: 1
- Pack Control**
- Max packet length: 1024
- Merge length: 1
- Idle Time: 20 (ms)
- Net Idle Time: 5 (ms)
- Latch: 10 (ms)
- Enable Match Packing:
- Match 2 Bytes Sequence: Yes No
- Send Frame Only: Yes No
- Match Byte: 0x31 0x32 (Hex)

A "Submit" button is located at the bottom of the form.

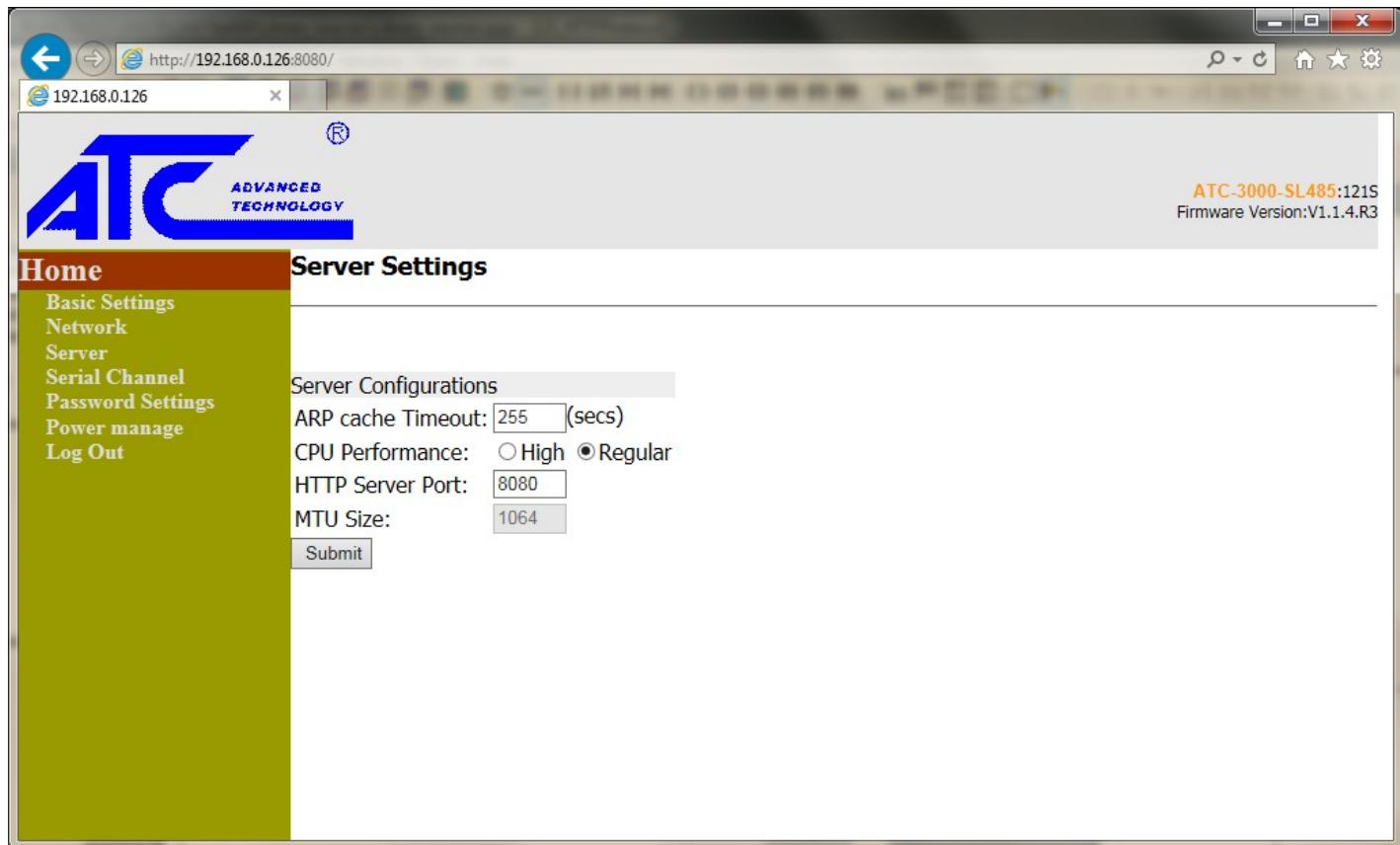
Nella connessione TCP/IP ho definito come porta remota la porta HTTP (80) e come remote host il DNS del sito.



Collegando la porta seriale COM0 del sistema SlimLine alla porta seriale del convertitore ATC-3000 i dati inviati in uscita dalla FB **HTTPGetPage** raggiungono il server di Altervista che gestisce la richiesta ritornando la pagina richiesta.

Il contenuto della pagina ricevuta dall'ATC-3000 sono passati via porta seriale allo SlimLine che li interpreta.

Attenzione occorre modificare la porta HTTP del convertitore (Esempio 8080).



DEPRECATO

7.18 Functions and FBs for Hw Group products (eHwGSpLib)

The HW Group company in the Czech Republic (<http://www.hw-group.com>) produces networking devices and products for remote control, monitoring and data management.

All Hw Group's products have connectivity over Ethernet with TCP/IP, UDP, SNMP protocol. To facilitate the connection of these products with the LogicLab development environment, specific functions and function blocks are given.

DEPRECATED

7.18.1 STESnmpAcq, STE thermometer acquisition over SNMP

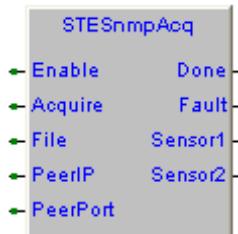
Type	Library
FB	eHwGSpLib_B000

This function block manages the acquisition of the values of the two temperature probes connected to the IP STE thermometer. The connection between the thermometer and the SlimLine is made over an ethernet network using SNMP protocol.

You must pass to FB the data stream indicated by the **File** parameter, previously opened by the [Sysfopen](#) function as a socket, and the socket must have been set in listening status using the [SysSktListen](#) function.

On the rising edge of the **Acquire** command, a SNMP read is made from the STE thermometer identified by **PeerIP** address and port defined by **PeerPort** (By default, the SNMP port is 161). If the **Acquire** command stay active, the read is done cyclically.

The **Done** output is activated for a program loop at the end of the acquisition of the two temperature values.



Enable (BOOL)	Enables function block.
Acquire (BOOL)	On the rising edge of command, start the acquisition of temperature from STE thermometer. On the rising edge of command, the acquisizione termometro STE. If command stay active, the read is done cyclically.
File (FILEP)	Stream returned by Sysfopen function.
PeerIP (STRING[15])	IP address of IP thermometer.
PeerPort (UINT)	Port used for connection (By default, the SNMP port is 161).
Done (BOOL)	Active for a program loop at the end of read.
Fault (BOOL)	Active for a program loop if there is an error.
Sensor1 (REAL)	Temperature value from sensor 1 (°C).
Sensor2 (REAL)	Temperature value from sensor 2 (°C).

Error codes

If an error occurs, the **Fault** output is activated and [SysGetLastError](#) can detect the error code.

- 10013010 **File** value not defined.
- 10013050 Execution timeout.
- 10013060 Error on read management.
- 10013100 Error controlling the IP address of STE device.
- 10013120 Error receiving data from STE device.
- 10013200~1 Error reading sensor 1.
- 10013300~1 Error reading sensor 2.

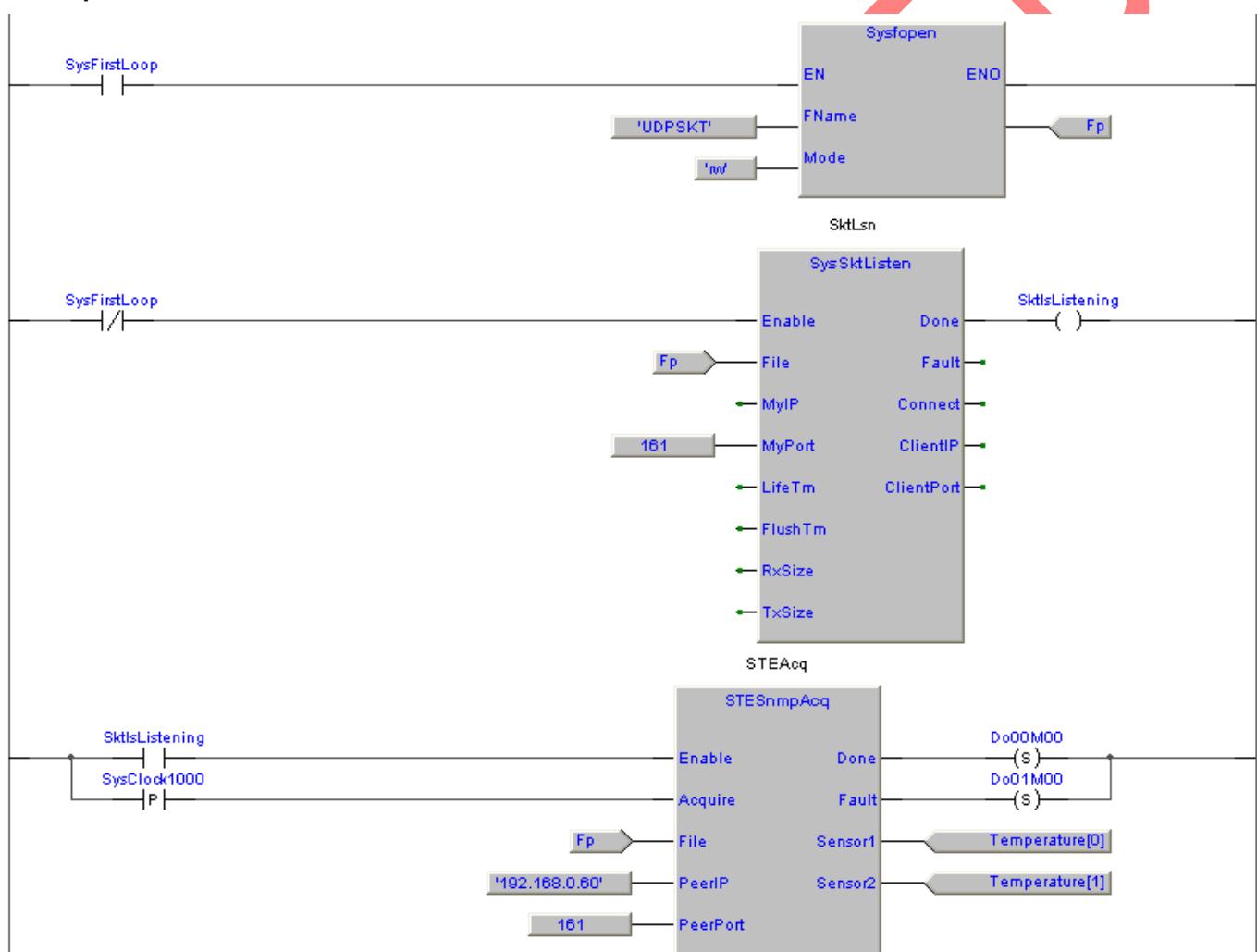
Examples

In the example, every second is managed the acquisition of the two values of temperature from a STE thermometer. The value of temperature in Celsius degrees is returned in the variables **Temperature[0]** and **Temperature[1]**. The **Do00M00** logic output is activated at the first acquisition, while the **Do01M00** logic output is activated when an error detected.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	SktIsListening	BOOL	Auto	No	FALSE	..	Socket is listening
2	Temperature	REAL	Auto	[0..1]	2(0)	..	Temperature
3	Fp	FILEP	Auto	No	0	..	UDP socket
4	STEAcq	STESnmpAcq	Auto	No	0	..	STE acquisition
5	SktLsn	SysSktListen	Auto	No	0	..	FB socket listen data

LD example



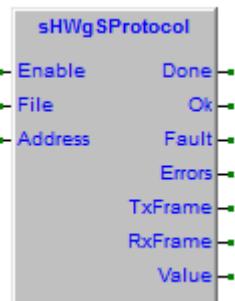
7.18.2 sHWgSProtocol, HW group serial protocol

Type	Library
FB	eHwGSpLib_B000

This function block handles the reading of the HW group devices by using the RS485 ascii protocol. You must pass the pointer of the data stream of type serial port indicated by the **File** parameter previously opened by the [Sysfopen](#) function.

Activating the **Enable** input is read the value from the addressed device connected to the serial port defined, terminated the execution the **Done** output is activated. If the command is successful the **Ok** output is set, otherwise the **Fault** output is set.

To repeat the command it's necessary to deactivate and then activate the **Enable** input, function block has been designed to allow the cascade connection. In practice it is possible to connect to an the **Done** output of one FB to the **Enable** input of another FB and so on.



Enable (BOOL)	Enables function block.
File (FILEP)	Stream returned by Sysfopen function.
Address (STRING[1])	String that defines the device address.
Done (BOOL)	Active for a program loop at the end of data read.
Ok (BOOL)	Active on value read.
Fault (BOOL)	Active on error.
Errors (UDINT)	Number of errors. Incremented every new error occurs. When it reaches the maximum value, it restart from 0.
TxFrame (STRING[8])	Contain the frame sent to device. It can be used for debug purpose.
RxFrame (STRING[16])	Contain the frame received from device. It can be used for debug purpose.
Value (REAL)	Acquired value.

Error codes

If an error occurs, the **Fault** output is activated, the **Errors** value is incremented and [SysGetLastError](#) can detect the error code.

- 10032010 **File** value not defined.
- 10032050 Execution timeout.
- 10032060 Error on read management.
- 10032100 Fb used on fast or slow task.
- 10013200~1 Error in the communication protocol management.
- 10032300 Error on value reading.

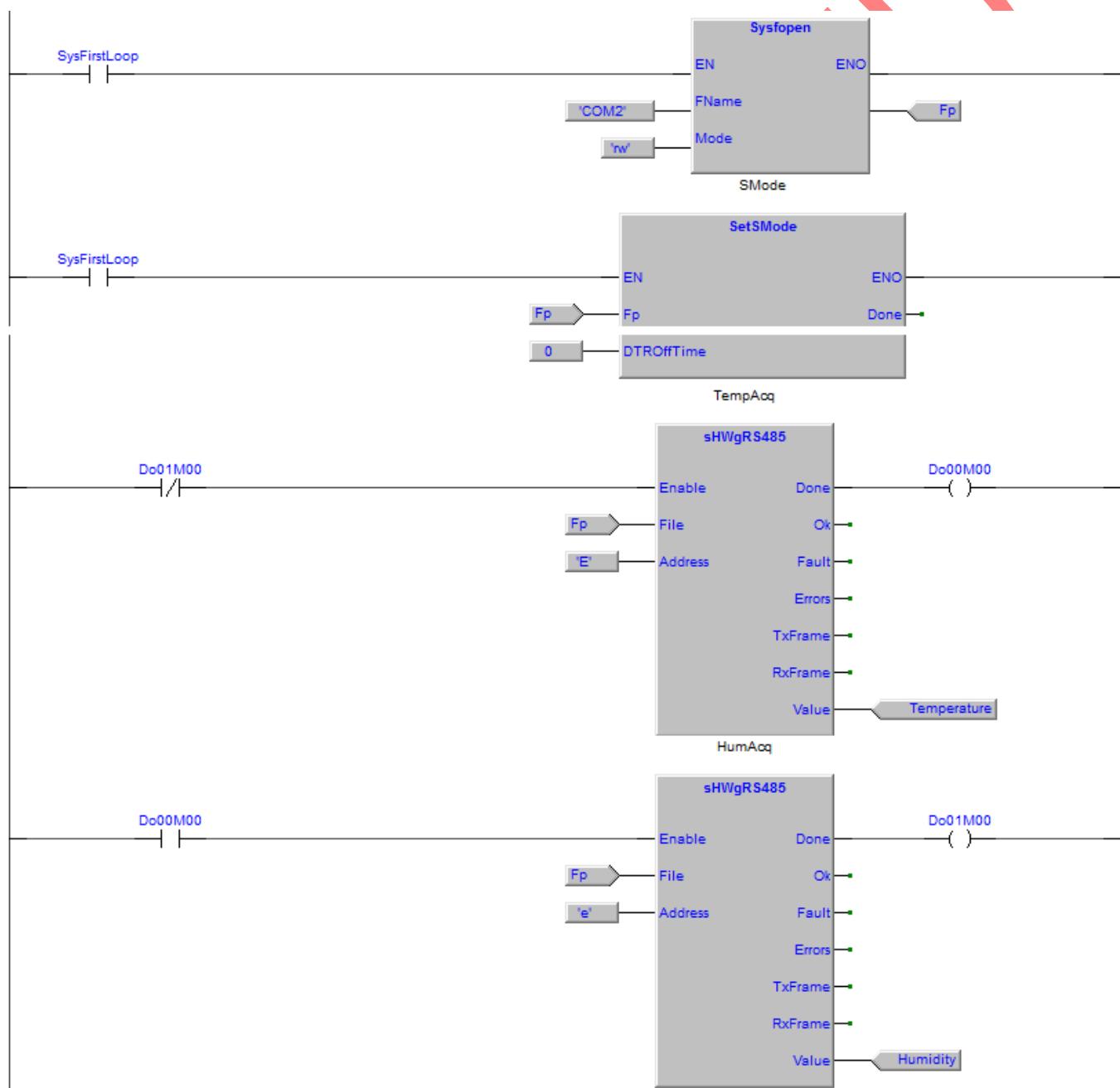
Examples

This example is the acquisition of the temperature and humidity from a HTemp device.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Humidity	REAL	Auto	No	0	..	Humidity value (%)
2	Temperature	REAL	Auto	No	0	..	Temperature value (°C)
3	Fp	FILEP	Auto	No	0	..	File pointer
4	SMode	SetSMode	Auto	No	0	..	FB set serial mode data
5	HumAcq	sHWgSProtocol	Auto	No	0	..	FB HTempBox data
6	TempAcq	sHWgSProtocol	Auto	No	0	..	FB HTempBox data

LD example (PTP126A000, LD_HTBoxRead)



7.19 Functions and FBs for NMEA protocol (eNMEALib)

Caution! To use the library it must be imported into your project. See the chapter [how to import libraries](#).

NMEA 0183 (or more commonly NMEA) is a data communication standard used primarily in marine and GPS satellite data communications. The entity that manages and develops the protocol is the National Marine Electronics Association.

This protocol is based on the principle that the source, also called talker, can only send data (sentences) and the receiver, also called listeners, can only receive them.

The **eNMEALib** library provides a series of functions and function blocks to handle the NMEA sentences. In practice it is possible to create programs with the LogicLab development environment that act as listeners of NMEA sentences.

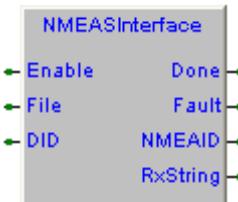
DEPRECATED

7.19.1 NMEASInterface, NMEA system interface

Type	Library
FB	eNMEALib_B000

This function block handles the interface to a device that sends NMEA sentences. This device must be connected to the I/O terminal defined in File. This function block is protected and require a code to unlock it (see [functions and function blocks protection](#)). It is possible to use it freely in test mode for 30 min.

FB receives NMEA sentences from the device, it checks the code by comparing it with the string defined in the **DID** and see if the sentence received is correct (Check the CRC). The **Done** output is activated for a program loop at each receiving NMEA sentence correct.



FB returns a **NMEAID** to be passed to the associated FB (FB NMEA sentences management). The output shows the **RxString** string received from the device. In this way you can show it, allowing you to view any errors.

Enable (BOOL)	Enables function block.
File (FILEP)	Stream returned by Sysopen function.
DID (STRING[2])	String that define the device prefix.
Done (BOOL)	Active for a program loop when a correct sentence is received.
Fault (BOOL)	Active for a program loop if there is an error.
NMEAID (UDINT)	NMEA ID to pass to the linked FBs.
RxString (STRING[82])	Contain the string received from device. It can be used for debug purpose.

Error codes

If an error occurs, the **Fault** output is activated, and [SysGetLastError](#) can detect the error code.

- 10017010 **File** value not defined.
- 10017020 Protected FB. The available time for demo mode is over.
- 10017050 Execution timeout.
- 10017070 Management case error.
- 10017100~4 Error receiving NMEA sentence.

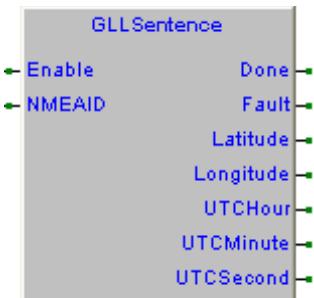
7.19.2 GLLSentence, Geographic Position sentence

Type	Library
FB	eNMEALib_B000

This function block performs the receipt of the GLL Geographic Position sentence. It connects to the **NMEAInterface** function block using the **NMEAID** variable.

The GLL sentence contains the information of latitude, longitude, time and fix. Example of sentence **\$IIGLL,4419.0173,N,00829.6653,E,084550.00,A,2*09**.

The FB checks the correctness of sentence fields and extracts the information of latitude, longitude and time. The **Done** output is activated for a program loop at each correctly received GLL sentence.



Enable (BOOL)	Enables function block.
NMEAID (UDINT)	NMEA ID supplied as output from NMEAInterface .
Done (BOOL)	Active for a program loop if a correct GLL sentence received.
Fault (BOOL)	Active for a program loop if there is an error.
Latitude (REAL)	Latitude value extracted from sentence. The value is expressed as a fraction of degrees. Positive values indicate north latitude, negative values indicate southern latitude.
Longitude (REAL)	Longitude value extracted from sentence. The value is expressed as a fraction of degrees. Positive values indicate north longitude, negative values indicate southern longitude.
UTCHour (USINT)	UTC hours extracted from sentence.
UTCMinute (USINT)	UTC minutes extracted from sentence.
UTCSecond (USINT)	UTC seconds extracted from sentence.

Error codes

If an error occurs, the **Fault** output is activated, and [SysGetLastError](#) can detect the error code.

- 10018010 **NMEAID** not defined.
- 10018020 **NMEAID** not correct.
- 10018100~2 Latitude value error.
- 10018200~2 Longitude value error.
- 10018300~2 UTC hour value error.

Examples

It is available a sample program Ptp123*000 that manages the interface to a navigation system with the interpretation of some NMEA sentences.

The example is managed the receive of a GLL sentence.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Fp	FILEP	Auto	No	0	..	File pointer
2	GLL	GLLSentence	Auto	No	0	..	GL sentence reception FB data
3	NMEAID	UDINT	Auto	No	0	..	NMEA interface ID
4	NMEARx	NMEAISInterface	Auto	No	0	..	NMEA device interface FB data
5	RxGPS	STRING	Auto	[32]		..	Rx data from GPS device
6	SMode	SetSMode	Auto	No	0	..	Serial mode FB data

LD example

DEPRECATE

7.19.3 MWVSentence, Wind Speed and Angle sentence

Type	Library
FB	eNMEALib_B000

This function block performs the receipt of the sentence MWV wind speed and angle. It connects to the **NMEAInterface** function block using the **NMEAID** variable.

The MWV sentence contains the information of speed and wind direction. An example sentence is **\$IIMWV,120.09,R,4.53,N,A*35**.

The FB checks the correctness of the sentence fields and extracts the information of speed and direction. The **Done** output is activated for a program loop at each correctly received MWV sentence.



Enable (BOOL) Enables function block.

NMEAID (UDINT) NMEA ID supplied as output from [NMEAInterface](#).

Done (BOOL) Active for a program loop if a correct MWV sentence received.

Fault (BOOL) Active for a program loop if there is an error.

WSpeed (REAL) Wind speed value (Nodes).

WPAngle (REAL) Polar angle value (Relative) of wind direction.

Error codes

If an error occurs, the **Fault** output is activated, and [SysGetLastError](#) can detect the error code.

10020010 **NMEAID** not defined.

10020020 **NMEAID** not correct.

10020100 Wind speed value error.

7.20 Functions and FBs for Power One inverter (ePowerOneLib)

Power One is a leading global manufacturer of power systems. Power One is right even in the field of alternative energy systems with applications for wind and solar inverters. Today, a convincing strategy in alternative energy can not be separated from the development of solutions for energy saving.

Aurora Photovoltaic Inverter Line, includes models for the network connection or isolated, with or without a transformer and designed for indoor and outdoor applications. All products are positioned in the range of solutions for design and construction technology at the top of the market and are characterized by high reliability, innovation and efficiency.

Aurora inverters

High conversion efficiency and ease of maintenance by allowing quick connection and disconnection of the photovoltaic modules. The scalable "Add-on" architecture of the system can cover a wide range of applications (up to 300kW on a single cabinet).

Also available without LV transformer for direct connection to a medium voltage system (with transformer MT).

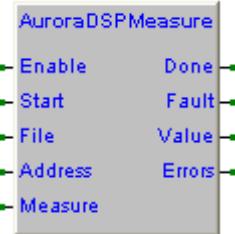
DEPRECATED

7.20.1 AuroraDSPMeasure, Aurora measure request to DSP

Type	Library
FB	ePowerOneLib_B000

This function block reads the measurements from the DSP of a Power One Aurora inverter, connected to the I/O device defined in **File**. This function block is protected and require a code to unlock it (see [functions and function blocks protection](#)). It is possible to use it freely in test mode for 30 min. It is used the [GetPolynomialCRC](#) function for CRC calculation of data frame to and from the inverter.

The connection to the inverters is RS485 multidrop. In **Address** you must define the inverter address with which you want to talk. In **Measure** you must specify the measure code of what you want to read (See measure codes).



By activating the **Start** input, the chosen measure is read. After the reading, the **Done** output will be activated for a program loop. If an error occurs, the **Fault** output will be activated for a program loop and the **Errors** value will be incremented.

Enable (BOOL)	Enables function block.
Start (BOOL)	Command to read the measure.
File (FILEP)	Stream returned by Sysfopen function.
Address (USINT)	Inverter address (Range from 0 to 255).
Measure (USINT)	Type of measure to read from inverter (See measure codes).
Done (BOOL)	Active for a program loop at the end of command execution.
Fault (BOOL)	Active for a program loop if there is an error.
Value (REAL)	Measure value read from inverter (It is in the relative measure unity).
Errors (UDINT)	Number of errors. Incremented every new error occurs. When it reaches the maximum value, it restart from 0.

Measure codes

In the **Measure** variable, you must define the Code of the measure to do from the inverter according to the table below.

Code	Description	Um
1	Grid Voltage (For three-phases systems is the mean)	V
2	Grid Current (For three-phases systems is the mean)	A
3	Grid Power (For three-phases systems is the mean)	W
4	Frequency (For three-phases systems is the mean)	Hz
5	Vbulk (For Inverter with more Bulk is the sum)	V
6	Ileak (Dc/Dc)	A
7	Ileak (Inverter)	A
21	Inverter Temperature	°C
22	Booster Temperature	°C
23	Input 1 Voltage (Input Voltage for single channel module)	V
25	Input 1 Current (Input Current for single channel module)	A
26	Input 2 Voltage (Input Voltage for single channel module)	V
27	Input 2 Current (Input Current for single channel module)	A
28	Grid Voltage (Dc/Dc)	V
29	Grid Frequency (Dc/Dc)	Hz

Code	Description	Um
30	Isolation Resistance (Riso)	
31	Vbulk (Dc/Dc)	V
32	Average Grid Voltage (VgridAvg)	V
33	VbulkMid	V
34	Power Peak	W
35	Power Peak Today	W
36	Grid Voltage neutral	V
37	Wind Generator Frequency	Hz
38	Grid Voltage neutral-phase	V
39	Grid Current phase r	A
40	Grid Current phase s	A
41	Grid Current phase t	A
42	Frequency phase r	Hz
43	Frequency phase s	Hz
44	Frequency phase t	Hz
45	Vbulk +	V
46	Vbulk -	V
47	Supervisor Temperature	°C
48	Alim. Temperature	°C
49	Heat Sink Temperature	°C
61	Grid Voltage phase r	V
62	Grid Voltage phase s	V
63	Grid Voltage phase t	V

Error codes

If an error occurs, the **Fault** output is activated, and [**SysGetLastError**](#) can detect the error code.

- 10030010 **File** value not defined.
- 10030020 Protected FB. The available time for demo mode is over.
- 10030050 Execution timeout.
- 10030070 Management case error.
- 10030100 CRC error on answer from Aurora inverter.
- 10030200 Error receiving the “Transmission state” from Aurora inverter.
- 10030251 Error from Aurora inverter: “Command is not implemented”.
- 10030252 Error from Aurora inverter: “Variable does not exist”.
- 10030253 Error from Aurora inverter: “Variable value is out of range”.
- 10030254 Error from Aurora inverter: “EEeprom not accessible”.
- 10030255 Error from Aurora inverter: “Not Toggled Service Mode”.
- 10030256 Error from Aurora inverter: “Can not send the command to internal micro”.
- 10030257 Error from Aurora inverter: “Command not Executed”.
- 10030258 Error from Aurora inverter: “The variable is not available, retry”.

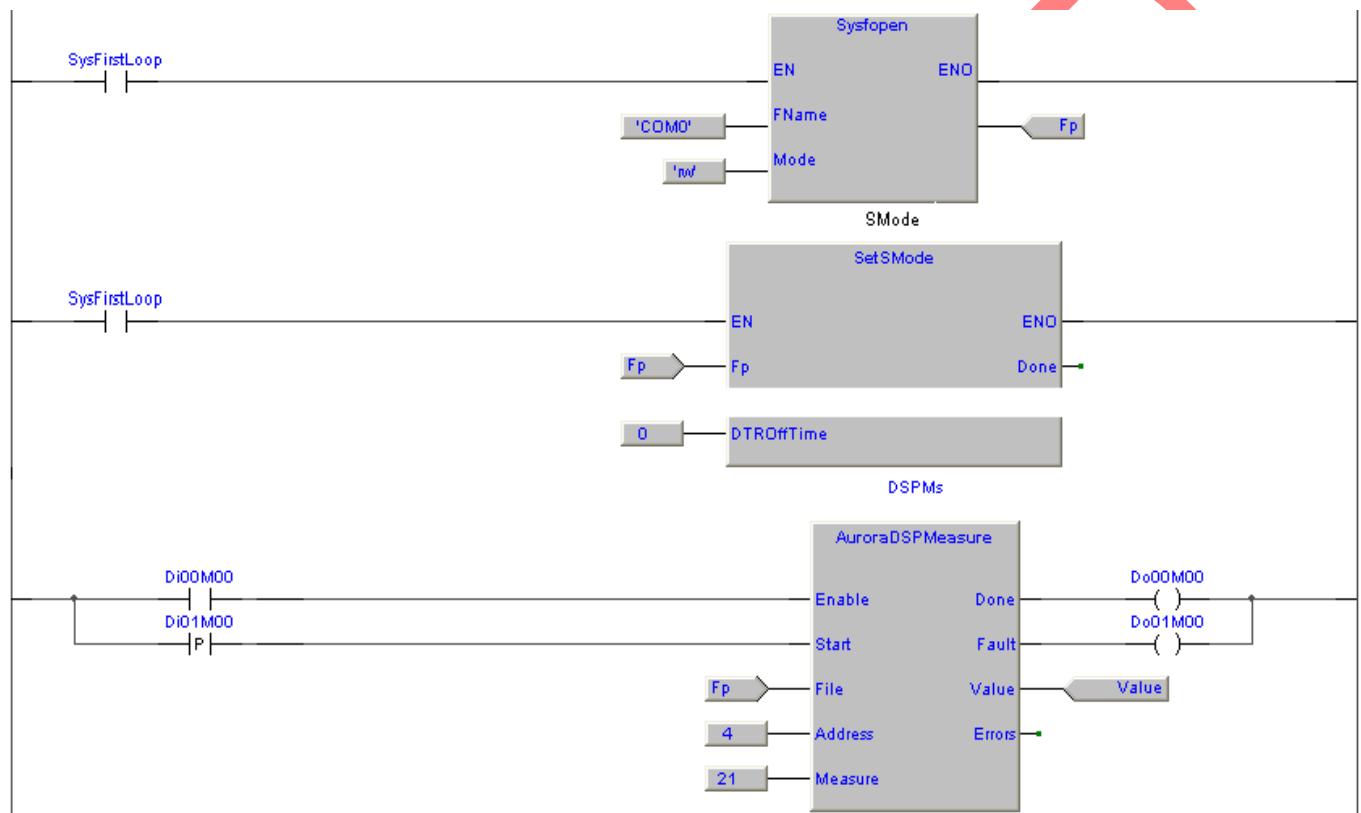
Examples

It is executed the read of the measure 21 (Inverter Temperature) from inverter with address 4. The return value is transferred to the **Value** variable. By default the serial port must be set to **19200,n,8,1**.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Fp	FILEP	Auto	No	0	..	Terminal I/O file pointer
2	SMode	SetSMode	Auto	No	0	..	FB set serial mode
3	DSPMs	AuroraDSPMeasure	Auto	No	0	..	FB Aurora DSP measure
4	Value	REAL	Auto	No	0	..	Value read from inveter

LD example

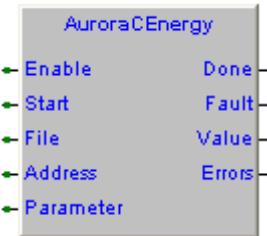


7.20.2 AuroraCEnergy, Aurora cumulated energy reading

Type	Library
FB	ePowerOneLib_B000

This function block reads the generated energy by a Power One Aurora inverter, connected to the I/O device defined in **File**. This function block is protected and require a code to unlock it (see [functions and function blocks protection](#)). It is possible to use it freely in test mode for 30 min. It is used the [GetPolynomialCRC](#) function for CRC calculation of data frame to and from the inverter.

The connection to the inverters is RS485 multidrop. In **Address** you must define the inverter address with which you want to talk. In **Parameter** you must specify the parameter to read (See parameter codes).



By activating the **Start** input, the chosen parameter is read. After the reading, the **Done** output will be activated for a program loop. If an error occurs, the **Fault** output will be activated for a program loop and the **Errors** value will be incremented.

Enable (BOOL)	Enables function block.
Start (BOOL)	Command to read the parameter.
File (FILEP)	Stream returned by Sysfopen function.
Address (USINT)	Inverter address (Range from 0 to 255).
Parameter (USINT)	Parameter code to read (See parameter codes).
Done (BOOL)	Active for a program loop at the end of command execution.
Fault (BOOL)	Active for a program loop if there is an error.
Value (UDINT)	Parameter value read from inverter (It is in the relative measure unity).
Errors (UDINT)	Number of errors. Incremented every new error occurs. When it reaches the maximum value, it restart from 0.

Parameter codes

In the **Parameter** variable, you must define the Code of the parameter to read from the inverter according to the table below.

Code	Description	Um
0	Daily energy	Kw
1	Weekly Energy	Kw
3	Month Energy (Energy from the first day of current calendar month)	Kw
4	Year Energy (Energy from the first day of current calendar year)	Kw
5	Total Energy (Total lifetime)	Kw
6	Partial Energy (Cumulated since reset)	Kw

Error codes

If an error occurs, the **Fault** output is activated, and [SysGetLastError](#) can detect the error code.

- 10031010 **File** value not defined.
- 10031020 Protected FB. The available time for demo mode is over.
- 10031050 Execution timeout.
- 10031060 Wrong parameter code.
- 10031070 Management case error.
- 10031100 CRC error on answer from Aurora inverter.

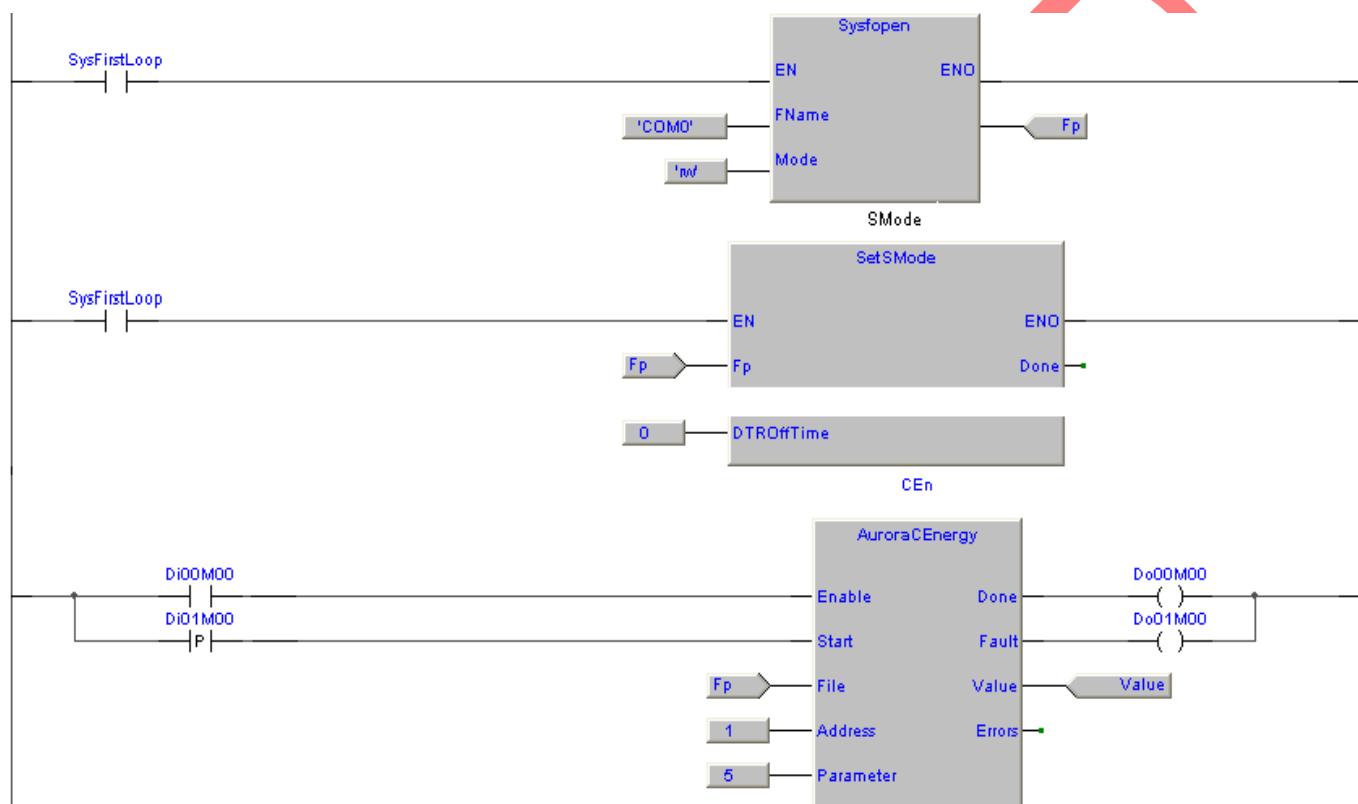
Example

It is executed the read of the total of the energy produced by the inverter with address 1. The return value is transferred to the **Value** variable. By default the serial port must be set to **19200,n,8,1**.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Fp	FILEP	Auto	No	0	..	Terminal I/O file pointer
2	SMode	SetSMode	Auto	No	0	..	FB set serial mode
3	Value	UDINT	Auto	No	0	..	Value read from inveter
4	CEn	AuroraCEnergy	Auto	No	0	..	FB Aurora cumulated energy

LD example



7.21 Functions and FB to manage logs (eLogLib)

This library provides a series of functions and function blocks for managing logs.

Sends notifications to a Syslog server

SYSLOG (System Log) is a protocol belonging to the Internet protocol suite used to transmit information through a simple network log. The protocol has been standardized by IETF.

Generally it is used over UDP through port 514, in particular applications where monitoring is essential, or certain events can trigger actions on the part of the SYSLOG server, a TCP connection and/or encryption.

The client sends a text message, up to 1024 characters, to the server, commonly referred to as "syslogd", "syslog daemon" or "syslog server". The protocol simplicity means that the server can handle messages from a wide range of machines as computers, printers, networking equipment, machinery, etc.. The server can simply record the event, or reacting to particular levels of severity calling programs, sending e-mail, etc.

A notification message sent to a Syslog Server begins with a plant indicator **Facility** and a gravity code **Severity**. The following tables showing the assigned codes.

Facility codes

- 0 Kernel messages
- 1 User-level messages
- 2 Mail system
- 3 System daemons
- 4 Security/authorization messages
- 5 Messages generated internally by syslogd
- 6 Line printer subsystem
- 7 Network news subsystem
- 8 UUCP subsystem
- 9 Clock daemon
- 10 Security/authorization messages
- 11 FTP daemon
- 12 NTP subsystem

Severity codes

- 0 Emergencies Sistema inutilizzabile
- 1 Alerts Richiede intervento immediato
- 2 Critical Condizioni critiche
- 3 Errors Condizione d'errore
- 4 Warnings Condizioni di warning
- 5 Notifications Condizioni di anomalia non critici (bugs)
- 6 Informational Messaggi informativi
- 7 Debugging Messaggi di debug

The message continues with an indication of the date and time, the name of the device that sent the message **Hostname** and message text **Message**.

Syslog server

A SYSLOG server is a central point where to get all the error messages of the various hardware devices and software on a network such as routers, switches, servers, printers, etc for centralized control of the errors of the equipment.

SYSLOG is particularly common in unix and consequently under linux, in windows there are some freeware and/or commercial programs to manage syslog server. The server can select (filter) incoming messages based on various criteria, each selection is at least one share.

In practice this is to establish selection criteria and the action to take depending on the source and type of message

The selection criterion can be for example:

Priority, IP address of the sender of the message, Hostname, text (or part of it) of the message, time interval of one or more days of the week.

The actions can be for example:

None (ignore the message), View it in the monitoring program, Send it to another Syslog server, Issuing an alarm sound, run a program, send an e-mail, Save the message in a database (eg MySQL), Save the message in a log file, run a script, etc.

For my needs I used **Syslog Watcher** (Site <http://www.snmpsoft.com>), an excellent free program, below is a screenshot with the display of notifications sent by a system SlimLine.

The screenshot shows the 'Syslog Watcher - Local Syslog Server' application window. The interface includes a toolbar with icons for Start Server, Stop Server, Status, Reload, Filter, Find, Search, Import, Export, Delete, Reports, Storage, Settings, Vendor Pack, Help, and Info. Below the toolbar is a menu bar with options like Show, Any Severity, from, All Sources, last, 1000, messages, Update every, 10, seconds, Updated at 27/01/2012 09:31:35, AutoScroll, and a help icon. The main area has tabs for Last 1000 Syslogs, Syslogs for Period, Import/Search Results (0), Sources (2), Server Log, and Backups. The 'Last 1000 Syslogs' tab is selected, displaying a table of log entries. The columns are Received, Source IP, Source Na..., Facility, Severity, Timestamp, Tag, Origin, and Message. The log entries are as follows:

Received	Source IP	Source Na...	Facility	Severity	Timestamp	Tag	Origin	Message
27/01/2012 09:15:56.438	192.168.0....		mail	Alert	Feb 27 09:16:15	0xBE421217	Antifurto	Allarme da sensore perimetrale
27/01/2012 09:15:10.375	192.168.0....		mail	Alert	Feb 27 09:15:28	0xBE421217	Caldaia	Bruciatore in blocco
27/01/2012 09:10:49.423	192.168.0....		syslogd	Warning	Feb 27 09:11:08	0xBE421217	Caldaia	Messaggio di prova
27/01/2012 09:13:50.574	192.168.0....		mail	Warning	Feb 27 09:14:08	0xBE421217	Caldaia	Messaggio di prova
27/01/2012 09:17:51.757	192.168.0....		mail	Alert	Feb 27 09:18:10	0xBE421217	Supervisore	Accesso da porta di servizio
27/01/2012 09:28:32.565	192.168.0....		secur/auth	Notice	Feb 27 09:28:51	0xBE421217	Supervisore	Ingresso operatore Nr. 124288
27/01/2012 09:28:54.488	192.168.0....		secur/auth	Notice	Feb 27 09:29:12	0xBE421217	Supervisore	Ingresso operatore Nr. 233567

Below the table is a 'Message View' pane showing a single entry:

```
Warning / syslogd | (192.168.0.125)
venerdì 27 gennaio 2012 09:13:20.853
Messaggio di prova
```

At the bottom of the window are status indicators: For Help, press F1; Service: Started (4.3.0); Tot: 7; Dsp: 7; Flt: 0; Sel: 0; UDP: 514; TCP: 1468; IPv4; IPv6; Ver: 4.3.0.

As you can see the various notices are divided by color according to their importance and in the message text can include any information such as code operator who has had access to entry detected by an RFID reader.

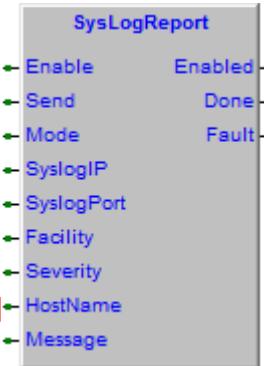
7.21.1 SysLogReport, send a report to Syslog server

Type	Library
FB	eLogLib_B000

This function block sends a notification messages to a Syslog server whose IP address is defined in **SyslogIP** and the port in **SyslogPort**. This function block is protected and require a code to unlock it (see [functions and function blocks protection](#)). It is possible to use it freely in test mode for 30 min.

It's possible to set the **Facility** code, the **Severity** and the name of the host system **HostName**.

By activating the **Send** input a notification is sent to a Syslog server, executed the sending the **Done** output is set for a program loop, if an error is detected the **Fault** output is set for a program loop.



Enable (BOOL)	Enables the function block.
Send (BOOL)	Send a Syslog notification.
Mode (USINT)	Operative mode, 0:UDP.
SyslogIP (STRING[16])	Syslog server IP address.
SyslogPort (UINT)	Port used by the Syslog server.
Facility (USINT)	Plant code.
Severity (USINT)	Gravity code.
HostName (STRING[32])	Name of the host system who sends the message.
Message (STRING[160])	Descriptive text of the notification message.
Enabled (BOOL)	Active if the FB is enabled.
Done (BOOL)	Active for a loop after sending the notification message.
Fault (BOOL)	Active for a loop on error executing the command.

Error codes

If an error occurs, the **Fault** output is activated, and [**SysGetLastError**](#) can detect the error code.

- 10034020 Protected FB. The available time for demo mode is over.
- 10034080 FB used in a fast or slow task.
- 10034100 Syslog server not reachable, it doesn't answer to a ping.

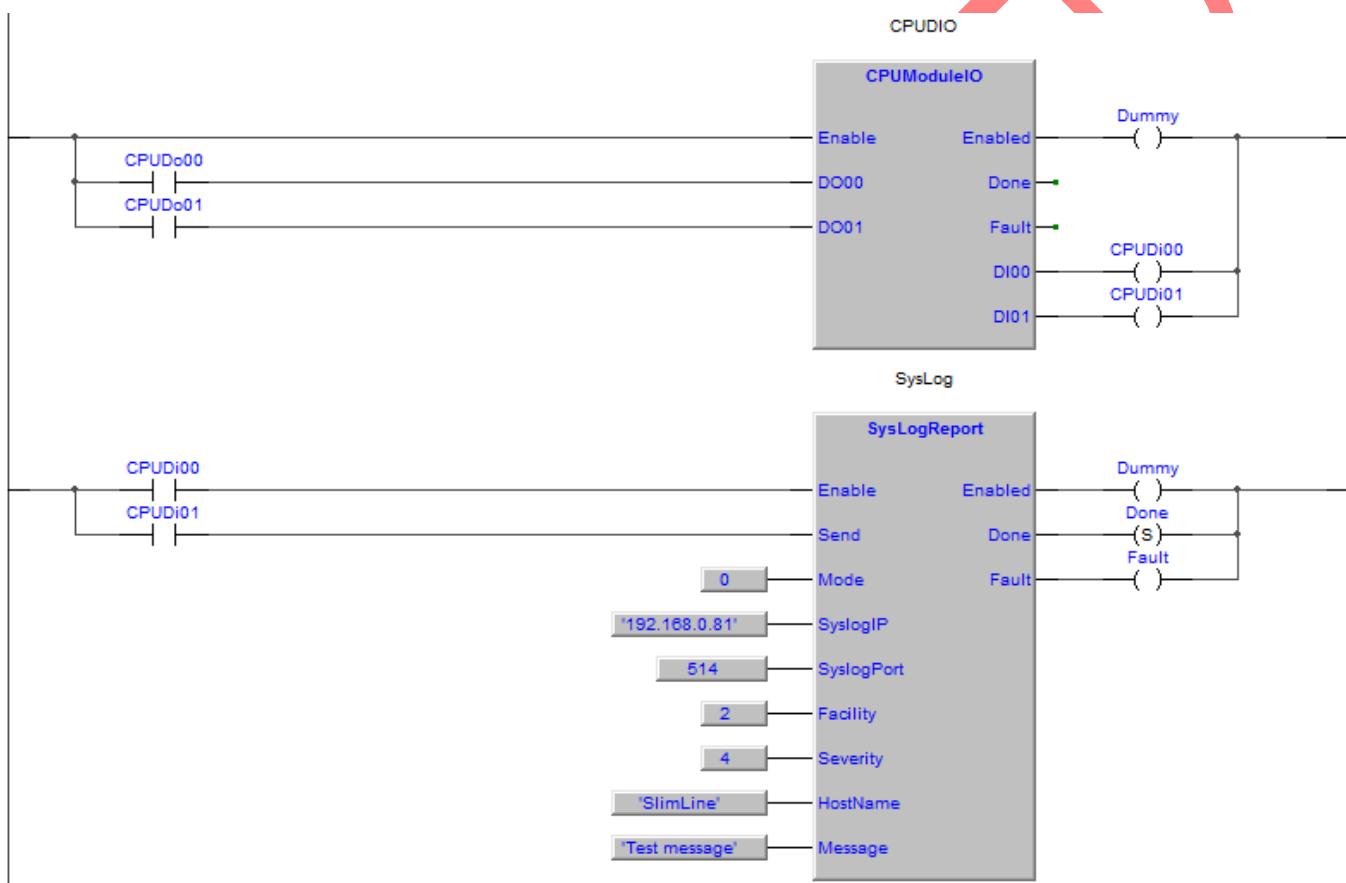
Examples

In the following example are acquired the logic I/O on CPU module, using the two inputs you can send a notification message to the syslog server with IP 192.168.0.81 on port 514 UDP.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Done	BOOL	Auto	No	FALSE	..	Syslog sent
2	Dummy	BOOL	Auto	No	FALSE	..	Dummy variable
3	Fault	BOOL	Auto	No	FALSE	..	Syslog error
4	CPUDIO	CPUModuleIO	Auto	No	0	..	FB CPU I/O management
5	SysLog	SysLogReport	Auto	No	0	..	FB SysLogReport

LD example



7.21.2 StringToFile, store string to a log file

Type	Library
FB	eLogLib_B000

This function block logs the **StringToLog** string into the file **Filename**. Each line is terminated with CR-LF. When the **MaxRowsInFile** lines are reached in the file, if **Circular** is set, the new log string are written starting from the beginning so overwriting the existing lines. If **Circular** is not set the string is not written in the file and an error occurs.

The variable **MaxRowsLen** allows to limit the maximum length of each line in the log file. When **Circular** is set, if the length of **StringToLog** is less than **MaxRowsLen**, some space characters are added to the **StringToLog** to have a length equal to **MaxRowsLen**. In **RowIndexPtr** must enter the address of the variable used as an index of the next row in which to log. Normally this variable must be buffered to allow you to write in the right place even after a power-off system. If you do not use a variable buffer at each switch-off, it will restart to write logs from the beginning of the file. So you can choose the desired behavior.

This function block is protected and require a code to unlock it (see [function and function block protection](#)). It is possible to use it freely in test mode for 30 min.

By activating the **Write** input the **StringToLog** string is stored on the file and then the **Done** output is set for a program loop. In case of execution error the **Fault** output is set for a program loop.



Enable (BOOL)

Enables the function block.

Write (BOOL)

Write **StringToLog** to file.

StringToLog (STRING[160])

String to be written in the file.

Filename (STRING[32])

Filename and path where to write the logs (ie.: 'SDCard/MyFile.txt').

Circular (BOOL)

Indicates if to execute the circular logs or not.

MaxRowLen (USINT)

String to log length limit.

MaxRowsInFile (UDINT)

Number of log rows in the log file.

RowIndexPtr (@UDINT)

Pointer to the variable used as an index of the next line of log.

Enabled (BOOL)

Set if the function block is enabled.

Done (BOOL)

Set for a program loop when the log string is written in the file.

Fault (BOOL)

Set for a program loop if an error occurs.

Error codes

If an error occurs, the **Fault** output is activated, and [SysGetLastError](#) can detect the error code.

10035020 Protected FB. The available time for demo mode is over.

10035080 FB used in a fast or slow task.

10035100 The **MaxRowLen** value is too large.

10035110 Maximum number of log entries reached.

10035120 Error on opening file.

10035130 File positioning error.

10035140 File write error.

10035150 File close error.

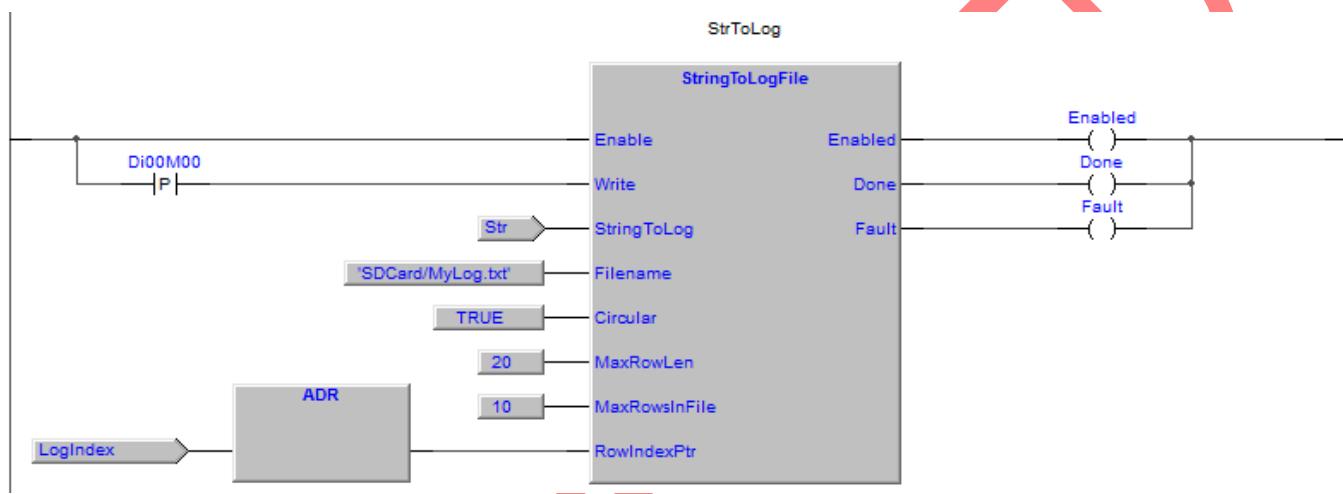
Examples

In the following example when the **Di00M00** input is set, the string present in the variable **Str** is saved in the file **SDCard/MyLog.txt**. Having **Circular** set, when 10 logs are saved the eleventh overwrites the first. The **LogIndex** variable is a variable mapped in the RETAIN area.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Str	STRING	Auto	[32]	Log string	..	String to log
2	StrToLog	StringToFile	Auto	No	0	..	FB String to log file
3	Done	BOOL	Auto	No	FALSE	..	FB done
4	Fault	BOOL	Auto	No	FALSE	..	FB fault
5	Enabled	BOOL	Auto	No	FALSE	..	FB enabled

LD example



7.21.3 FileMemoryDump, dump memory on file

Type	Library
FB	eLogLib_B000

This function block performs a memory dump of a memory area starting at address **MBufferPtr** for the number of bytes defined **MBufferSize** on a disk file **Filename**.

With a pulse command on **Write** input the file is created and the contents of the memory is written to the file. With a pulse command on **Read** input the file is read and its contents is written into the memory.

When the command is end the **Done** output is set for a program loop, in case of execution error the **Fault** output is set for a program loop.

The dump file on disk is an ascii file so you can edit it with any text editor, here's a sample file.

```
00000000: 00 AB 12 34 00 00 00 00 | 00 00 00 12 00 0F 0A CC
00000010: 02 00 00 00 00 00 00 00 | EF C0 DD 00 00 00 01 00
```



Enable (BOOL) Enables the function block.

Read (BOOL) Read dump file and store contents in memory buffer.

Write (BOOL) Write memory contents to a dump file.

Filename (STRING[32]) Filename and path of dump file (ie.: 'Storage/Dump.txt').

MBufferPtr (@USINT) Pointer to the memory buffer.

MBufferSize (UDINT) Size in bytes of buffer memory.

Enabled (BOOL) Set if the function block is enabled.

Done (BOOL) Set for a program loop at the command execution end.

Fault (BOOL) Set for a program loop on command execution error.

Error codes

If an error occurs, the **Fault** output is activated, and [SysGetLastError](#) can detect the error code.

- 10036070 Execution cases error.
- 10036100 FB used in a fast or slow task.
- 10036200 Error on file opening.
- 10036210 Error on file positioning, read command.
- 10036220 Error on file read command.
- 10036230 Error on file write command.
- 10036240~4 Data failure on file dump, read command
- 10036400 Error on file opening, write command.
- 10036410 Error on file positioning, write command.
- 10036420 Error on file write, write command.
- 10036430 Error on file close, write command.

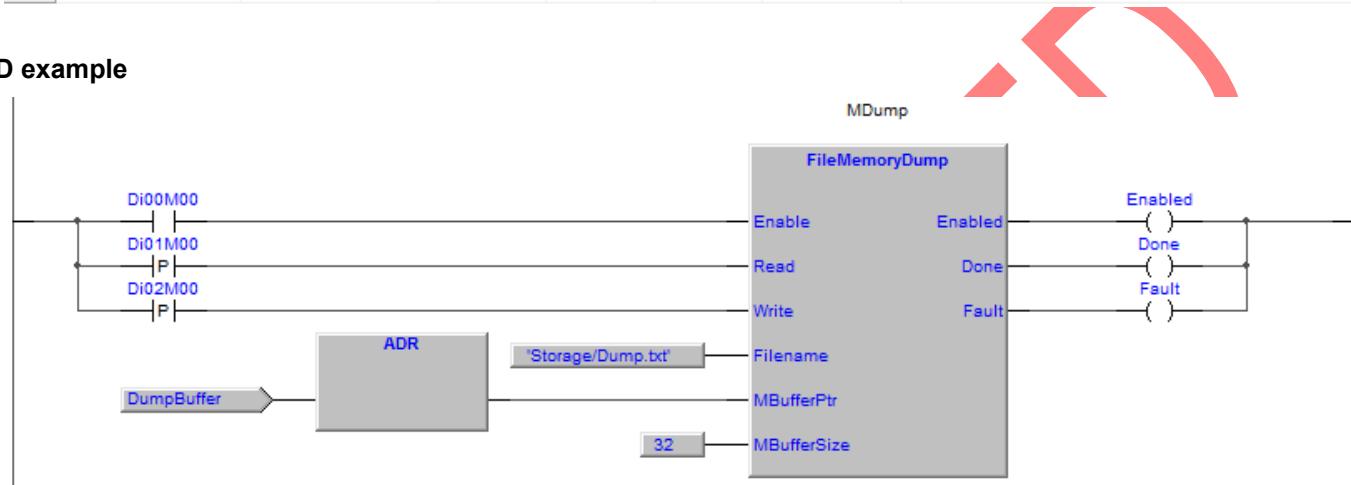
Example

In the following example on activation of the **Di02M00** input, the contents of the memory buffer **DumpBuffer**, is written on the **Storage/Dump.txt** file. After executing the write, by activating the **Di01M00** input, the file is read back and the data contained on it are transferred to the memory buffer.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	MDump	FileMemoryDump	Auto	No	0	..	FB File memory dump
2	DumpBuffer	USINT	Auto	[0..31]	32(0)	..	Data buffer
3	Enabled	BOOL	Auto	No	FALSE	..	FB enabled
4	Done	BOOL	Auto	No	FALSE	..	FB done
5	Fault	BOOL	Auto	No	FALSE	..	FB fault

LD example



7.22 Functions and FBs for multimaster communication (eMMasterDTxferLib)

Caution! To use the library it must be imported into your project. See the chapter [how to import libraries](#).

This library contains a series of functions and function blocks for the management of multimaster serial communication. The possibility to communicate on a single serial line RS422/485 multiple master devices allows to speed up the dialogue between the systems and to optimize the use of the serial line.

This possibility is useful in radio connections, where all the radiomodems share the same frequency so they can talk to each other minimizing the use of the band and increasing information transfer time.

In communication between different systems of twisted serial line and/or with radiomodems, typically are used packet protocols (Example Modbus) and a system that acts as a master cyclically communicates with all other systems on the network by exchanging information between them. As you can well understand this solution has the following limits:

- a) The communication is delegated to the master system, in case of failure all the network communication stop.
- b) The master must poll the various slaves systems to know if they have data to send to the master or to the other slaves. This implies an use of the communication channel even when the slaves have no information to be exchanged.
- c) The data exchange between two systems must pass from the master, this slows down the data sending from one system to another and increase the use of the communication channel.

Using this library, you can enable the systems to communicate directly with each other in a peer to peer connection. In this way a system sends data to another only when it is necessary, this ensures fast data transmission with minimum use of the communication channel.

Broadcast communication

When a device on a node of the network is in communication with many other devices and the data in exchange are the same for all devices the **BroadcastDataSend** function block can be used. It sends the packet data on broadcast, all the devices that have as destination node **ToNode** in the **DataTxferClient** function block the address of the node that sends the broadcast message will receive the messages without sending the acknowledge.

This means that the broadcast transmission allows to speed up the data transfer from one node to other nodes also allowing simultaneous reception of the data sent on all the nodes. But it is not certain the receipt of data by the recipient of the nodes, so it's suggested to follow at a broadcast transmission a peer to peer transmission. The following picture shows a typical scenario of multi-master communication via radio modem.

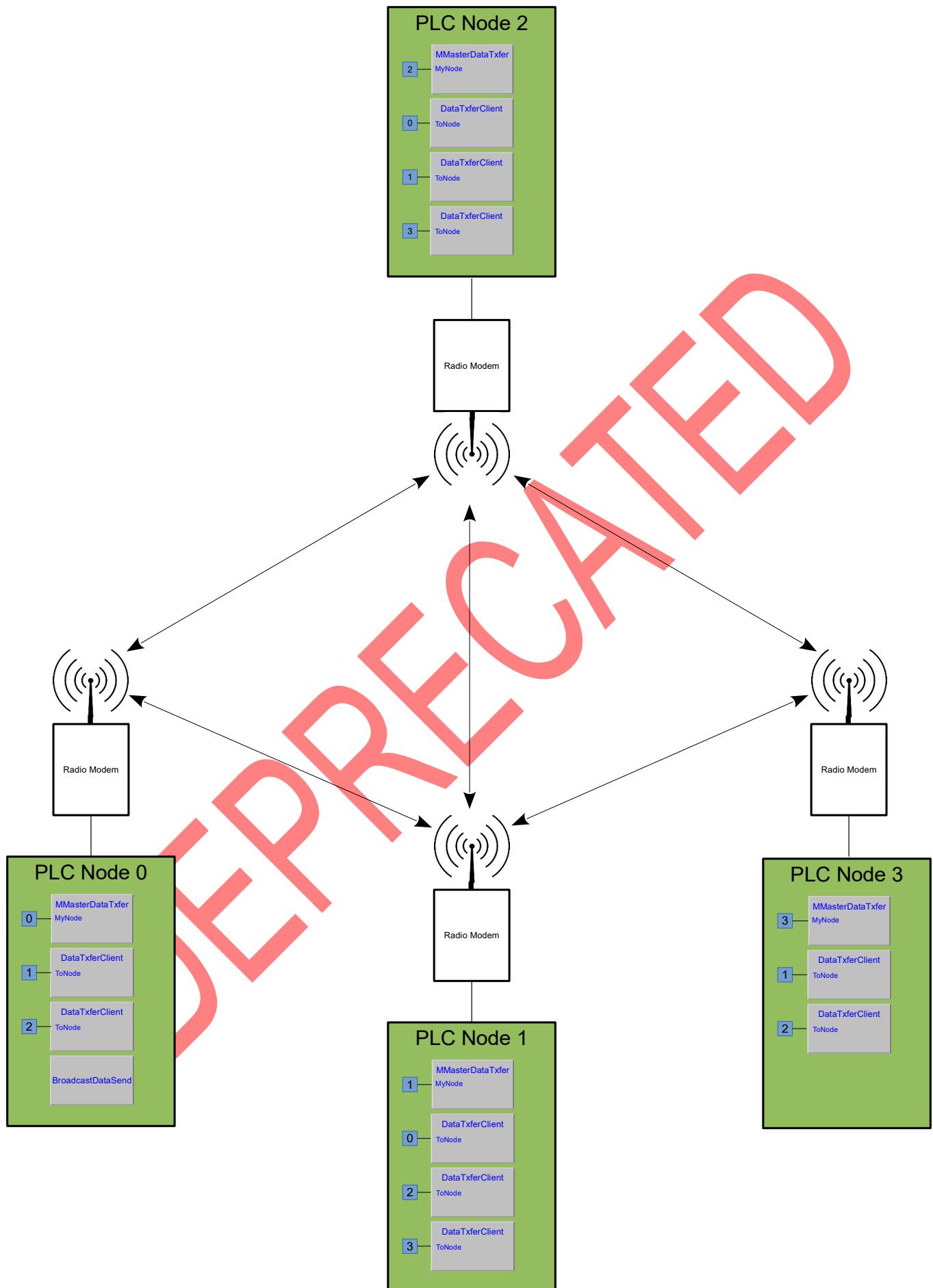
The PLC node 0 exchange data with the PLC node 1 and 2.

The PLC 1 node exchanges data with the PLC node 0, 2 and 3.

The PLC node 2 exchanges data with the PLC node 0, 1 and 3.

The PLC node 3 exchanges data with the PLC node 1 and 2.

In the PLC node 0 is also managed the FB broadcast sending, activating the transmission the data will be received only by the PLC node 1 and 2, and not by the PLC node 3 because it has no active exchange with the PLC of the node that sent the broadcast message.



7.22.1 MMasterDataTxfer, multimaster data transfer

Type	Library
FB	eMMasterDTxferLib_C000

This function block manage the interface with the I/O terminal defined in File to manage the multimaster communication on multidrop network. This function block is protected and require a code to unlock it (see [functions and function blocks protection](#)). It is possible to use it freely in test mode for 30 min.

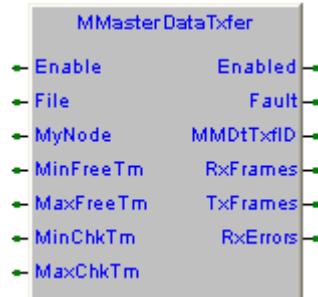
The function block acts as a server, managing the communication on the I/O device, some function blocks clients are then connected to it, they manage the data exchange between systems. The FB returns a **MMDtTxID** to be passed to the FB client (Example [DataTxferClient](#)).

In **MyNode** the system node number must be defined, on a network the node number must be unique. All messages with have as a destination node the value of **MyNode** will be received and then passed to the FB clients for verification.

The multimaster communication is based on the control of the communication channel free and the collisions management, the parameters that controls this operation are set in **MinFreeTm** and **MaxFreeTm**. A small time will be set for communications over the serial line and higher time in case of communication via radio modem.

In **MinChkTm** and **MaxChkTm** you can set a time for sending a control message to other network systems. The FB sequentially will enable the various FB clients to exchange data to the peer with which the FB dialogue checking if the communication is working properly. **If one or both values are set to 0, the check is not executed.**

In **RxErrors** is returned the reception error count, the FB continuously monitors the communication channel and if the received data is on error the counter is incremented. Errors may occur in case of collision on the communications channel. In case of execution error is set for for a loop the **Fault** output.



Enable (BOOL)

Enables function block.

File (FILEP)

Stream returned from **Sysfopen** function.

MyNode (USINT)

System node ID (Range from 0 to 250).

MinFreeTm (REAL)

Minimum waiting time for the communication channel free (S).

MaxFreeTm (REAL)

Maximum waiting time for the communication channel free (S).

MinChkTm (REAL)

Minimum waiting time to sending check frame to peer (S).

MaxChkTm (REAL)

Maximum waiting time to sending check frame to peer (S).

Enabled (BOOL)

FB enabled.

Fault (BOOL)

Active for a program loop if there is a management error.

MMDtTxID (UDINT)

Server ID to pass to FB clients (Example [DataTxferClient](#)).

RxFrames (UDINT)

Data frame received counter. All frames are counted regardless the node they are directed.

TxFrames (UDINT)

Data frame transmitted counter.

RxErrors (UDINT)

Data frame received error counter.

Error codes

If an error occurs, the Fault output is activated and [**SysGetLastError**](#) can detect the error code.

10040010 **File** not defined.

10040020 Protected FB. The available time for demo mode is over.

10040050 Execution timeout.

10040070 Wrong execution case.

7.22.2 DataTxferClient, Data transfer client

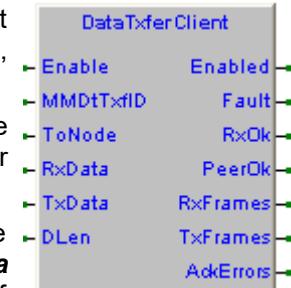
Type	Library
FB	eMMasterDTxferLib_C000

This function block exchanges data with another system on a communication channel. It connects to the **MMasterDataTxfer** function block that manages the communication device, must be passed the **MMDtTxflD** on output from the server function block.

The function block exchanges data with the system defined in **ToNode**. In practice, the value of **ToNode** must match the value of MyNode of the **MMasterDataTxfer** FB on the other system.

In **RxData** and **TxData** must be defined the address of data buffer that you want to exchange with the peer system. **DLen** defines the size in bytes of the exchange data buffer (The **RxData** and **TxData** buffers must have the same size). The FB checks whether there is a variation of the data in the **TxData** buffer and immediately transmit them to the peer that responds with the data in its **TxData** buffer, the received data will be transferred into the **RxData** buffer.

The **PeerOk** output is active if communication with the peer system is operational in the event of communication failure the output is disabled. In **RxFrames** and **TxFrames** are returned the count of data frames received and sent by the FB to the peer system, in **AckErrors** is returned the number of acknowledge errors. In case of execution error is set for a loop the **Fault** output.



Enable (BOOL) Enables function block.

MMDtTxflD (UDINT) Server ID on output from the **MmasterDataTxfer** FB.

ToNode (USINT) Node ID of the peer system on which to exchange data (Range from 0 to 250).

RxData (@USINT) Pointer to the buffer where the received data must be transferred.

TxData (@USINT) Pointer to the buffer where are the data to be transmitted.

DLen (UDINT) Number of exchanged bytes (Max 32).

Enabled (BOOL) FB enabled.

Fault (BOOL) Active for a program loop if there is a management error.

RxOk (BOOL) Active for a program loop when data is received from peer.

PeerOk (BOOL) On when data exchange with the peer system is ok.

RxFrames (UDINT) Rx frame counter.

TxFrames (UDINT) Tx frame counter.

AckErrors (UDINT) Peer system acknowledge errors

Error codes

If an error occurs, the **Fault** output is activated and [SysGetLastError](#) can detect the error code.

10041010 **MMDtTxflD** not defined.

10041020 Wrong **MMDtTxflD**.

10041050 **DLen** value out of range.

10041200 Data frames received from the peer system has incorrect length. Check **DLen** on peer system.

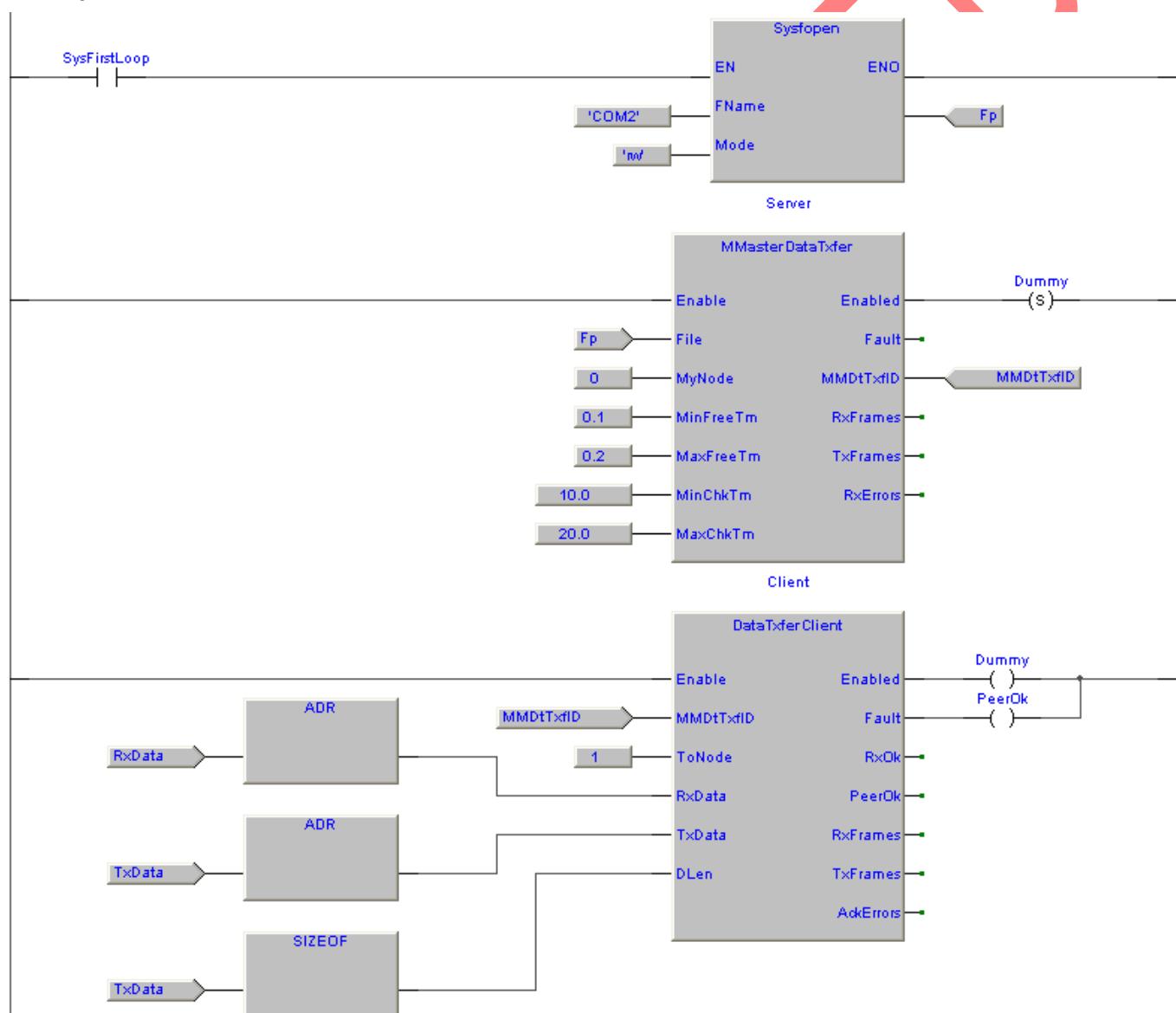
Examples

In the example is managed the exchange of 8 BOOL with the system peer node 1 (**MyNode=1**).

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Fp	FILEP	Auto	No	0	..	File pointer
2	MMDtTxID	UDINT	Auto	No	0	..	Multimaster data transfer ID
3	Client	DataTxferClient	Auto	No	0	..	Data transfer client
4	Server	MMasterDataTxfer	Auto	No	0	..	Multimaster data transfer
5	Dummy	BOOL	Auto	No	FALSE	..	Dummy variable
6	PeerOk	BOOL	Auto	No	FALSE	..	Peer system is Ok
7	RxData	BOOL	Auto	[0..7]	8(0)	..	Rx data from peer
8	TxDATA	BOOL	Auto	[0..7]	8(0)	..	Tx data to peer

LD example

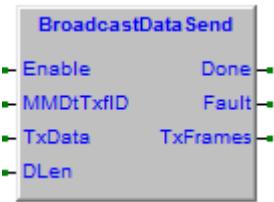


7.22.3 BroadcastDataSend, broadcast data send

Type	Library
FB	eMMasterDTxferLib_C000

This function block performs the broadcast data transmission. It connects to the **MMasterDataTxfer** function block that manages the communication device, must be passed the **MMDtTxflD** on output from the server function block.

The function block sends the data in the buffer pointed to by **TxData** with broadcast address 16#FF. All systems running the FB **DataTxferClient** who **ToNode** is equal to the value defined in the **Node** parameter of **MMasterDataTxfer**, receive the data sent.



In **TxData** need to define the address and in **DLen** the size in bytes of the data buffer you want to exchange with peer. To each activation of the **Enable** input the data is sent in broadcast, when sending is finished the **Done** output is activated and remains active until the deactivation of **Enable**. To make a new transmission must be disabled and then re-enable the **Enable** input.

Enable (BOOL)	FB enable
MMDtTxflD (UDINT)	Server ID on output from the MmasterDataTxfer FB.
TxData (@USINT)	Pointer to the buffer where are the data to be transmitted.
DLen (UDINT)	Number of exchanged bytes (Max 32).
Done (BOOL)	Active when data is sent, remain active until Enable is deactivated.
Fault (BOOL)	Active for a program loop if there is a management error.
TxFrames (UDINT)	Tx frame counter.

Codici di errore

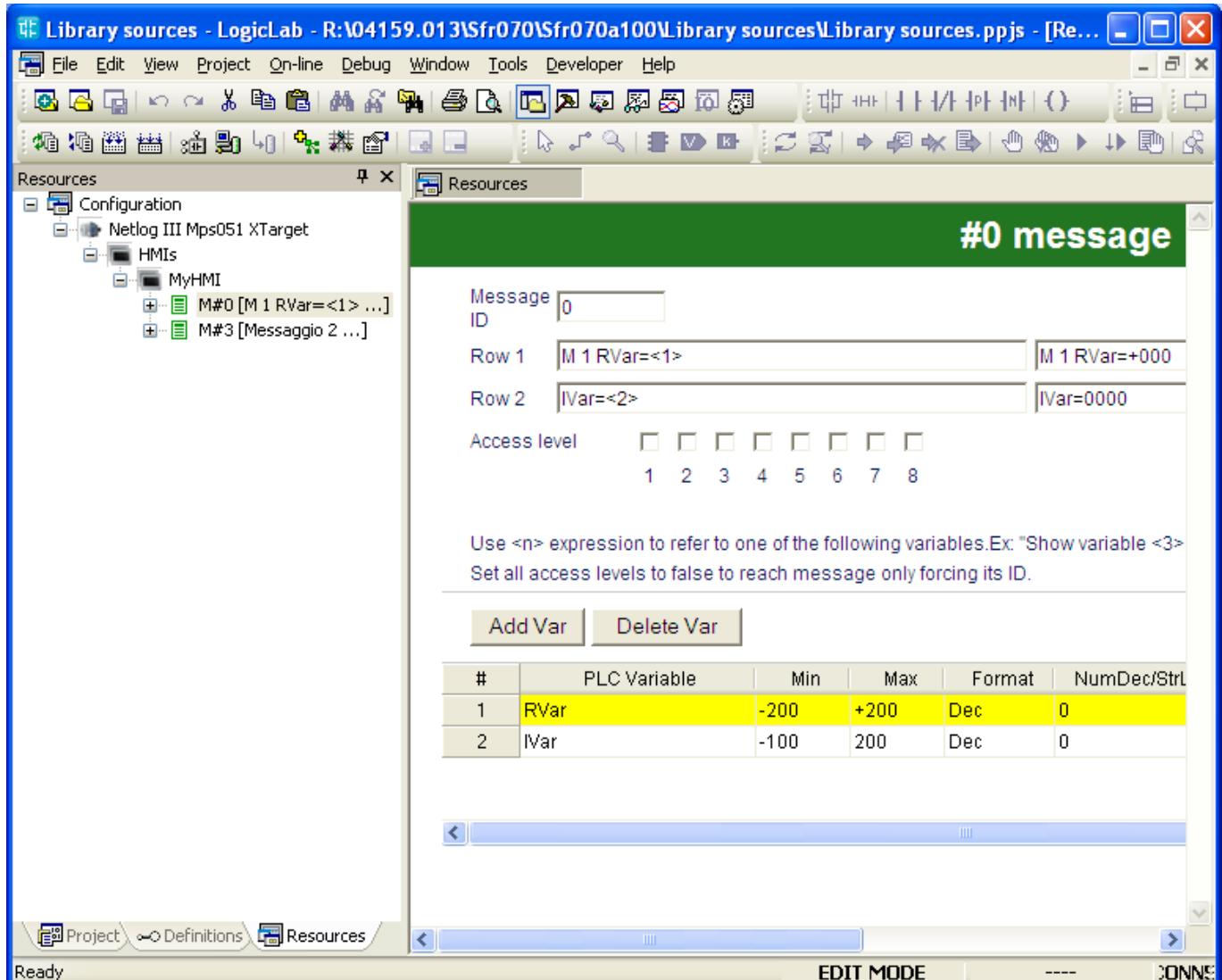
If an error occurs, the **Fault** output is activated and [**SysGetLastError**](#) can detect the error code.

- 10044010 **MMDtTxflD** not defined.
- 10044020 **MMDtTxflD** not correct.
- 10044050 **DLen** value out of range.

7.23 Human machine interface built-in library (eHMBuiltInLib)

Caution! To use the library it must be imported into your project. See the chapter [how to import libraries](#).

This library allows to manage the user interface directly from LogicLab. The **Resources** menu allows to configure the operator interfaces. As seen in the figure has been defined the **MyHMI** interface that includes two messages.



As you can see to each message is associated the text and the variables to be displayed. In the message can be displayed all global variables.

Compiling the project LogicLab creates and instantiates for each HMI a **HMBuiltInMessages** function block, named with the HMI name, in the example shown above **MyHMI**. This FB manages the messages and variables but to display them a terminal management FB must be connected to it.

7.23.1 HMIBuiltInMessages, HMI built in messages

Type	Library
FB	eHMIBuiltInLib_B000

This function block manages the messages as defined by LogicLab.

Even if it's reported its representation unlike the other function blocks it's automatically instantiated by LogicLab and must not instantiated by the user in its program. For the user it's important to refer only to the input/output FB's variables that may be used in its program.



The **Alevel** parameter configures the access level to messages, setting the bits of the variable allows you to display only the messages that have the same bit defined in **Access Level**.

CmdDisable (BOOL)	Terminal commands disable. By setting it all terminal operations are disabled.
ViewMID (UINT)	Displays the message with the defined ID.
ALevel (BYTE)	Access level pattern.
DEntryOk (BOOL)	Active for a loop at the end of variable data input.
MsgID (UINT)	Message ID of displayed message.
HMIBuiltInID (UDINT)	ID to pass to linked Fbs.

Error codes

If an error occurs, the Fault output is activated and [**SysGetLastError**](#) can detect the error code.

- 10046010 LogicLab messages table not supported.
- 10046020~2 Memory allocation not possible.
- 10046100 FB not executed in background (Back) task.
- 10046200 Message ID request on **ViewMID** not found.
- 10046100 Not enough space to display variable.

7.23.2 HMIBuiltInNetlog, Netlog HMI management

Type	Library
FB	eHMIBuiltInLib_B000

This function block manages the **NetlogIII** terminal (display and keyboard) integrated into the system, it must be executed on **Back** task. It connects to **HMIBuiltInMessages** management messages function block. The **HMIBuiltInID** of the function block must be connected to the input.

By activating the **Enable** input the **Enabled** output is activated and the terminal is managed displaying the messages defined in the LogicLab terminal. In the **File** input must be passed the terminal I/O stream (returned by the function **Sysfopen**).

The FB returns the state of all the terminal keys and this allows to use them in your program as a commands. The **SpyOn** input if active allows to spy the FB operations.

In case of execution error the **Fault** output is set for a program loop.



Enable (BOOL) FB enable command.

SpyOn (BOOL) Active allows to spy FB working.

TFlags	Description
16#00000001	Rx: Data received from terminal.
16#00000002	Tx: Display command sent.
16#00000004	Tx: Display data sent.
16#00000008	Tx: Input data command sent.

File (FILEP) Terminal I/O stream as returned by the **Sysfopen** function.

ALevel (USINT) Access Level pattern to messages.

HMIBuiltInID (UDINT) Management messages ID as returned by the **HMIBuiltInMessages** FB.

Enabled (BOOL) Active if the FB is enabled.

Ready (BOOL) Active if the Netlog terminal hardware is ready.

Fault (BOOL) Active for a program loop on error.

KeyUP (BOOL) UP key state, on terminal.

KeyDW (BOOL) DW key state, on terminal.

KeyLEFT (BOOL) LEFT key state, on terminal.

KeyRIGHT (BOOL) RIGHT key state, on terminal.

KeyFUN (BOOL) FUN key state, on terminal.

KeyENT (BOOL) ENT key state, on terminal.

Error codes

If an error occurs, the Fault output is activated and **SysGetLastError** can detect the error code.

10047010 **HMIBuiltInID** not defined.

10047020 **HMIBuiltInID** not correct.

10047100 FB not executed in background (Back) task.

Example

In this example a NetlgIII terminal is managed, the messages has been defined in the LogicLab program with the **MyHMI** name.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	HMI	HMIBuiltInNetlog	Auto	No	0	..	FB Netlog terminal
2	MsgID	UINT	Auto	No	0	..	Displayed message ID

ST example

```
(* Here open the terminal communication port. *)
IF (SysFirstLoop) THEN
    HMI.File:=Sysfopen('PCOM0.1', 'rw'); (* File pointer *)
END_IF;

(* Here manages the NETLOG III terminal. *)

MsgID:=MyHMI.MsgID; (* Displayed message ID *)
HMI.HMIBuiltInID:=MyHMI.HMIBuiltInID; (* ID gestione messaggi *)
HMI(Enable:=TRUE);
```

DEPRECATE

7.23.3 HMIPicoface, Picoface HMI management

Type	Library
FB	eHMIBuiltInLib_B200

This function block manages the Picoface terminal, the terminal can be connected to the I2C system bus extension or by serial both in RS232 FullDuplex or in RS485 HalfDuplex. The **CType** parameter specify the communication used.

The FB must be run in Task Back. It connects to **HMIBuiltInMessages** management messages function block. The **HMIBuiltInID** of the function block must be connected to the input.

By activating the **Enable** input it activates the **Enabled** output and the Picoface terminal is managed, viewing messages defined in the terminal from LogicLab. In the **File** parameter must be passed the I/O deice stream used to manage (Is the return of the **Sysopen** function).

Activating the input bits is possible to control the LEDs and outputs on the terminal, in the outputs are returned to the state of the keys and the terminal input. This allows to use them as commands in our program.

It's provided the management of a One Wire device, the FB in **OneWireData** returns an array with the data reported below.



Where:

DP: Device Protocol.

DF: Device family.

The **SpyOn** input if active allows to spy the FB operations.

If an error running is activated for a loop the **Fault** output.

HMIPicoface	
Enable	Enabled
CType	Ready
SpyOn	Fault
File	OneWireOk
HMIBuiltInID	OneWireTrig
FLed0	OneWireData
FLed1	FKey0
FLed2	FKey1
FLed3	FKey2
FLed4	FKey3
FLed5	FKey4
FLed6	FKey5
Backlight	FKey6
Out0	FKey7
Out1	FKey8
	FKey9
	KeyFUN
	KeyESC
	KeyCLR
	KeyENT
	KeyUP
	KeyDW
Inp0	
Inp1	

Enable (BOOL)

FB enable command.

CType (BOOL)

Communication type. **FALSE**: Full duplex (Default), **TRUE**: Half duplex

SpyOn (BOOL)

Active allows to spy FB working.

TFlags	Description
16#00000001	Rx : Data received from terminal.
16#00000002	Tx : Data sent to terminal.
16#10000000	Wn : Warning message (On communication error).

16#00000001 **Rx**: Data received from terminal.

16#00000002 **Tx**: Data sent to terminal.

16#10000000 **Wn**: Warning message (On communication error).

File (FILEP)

Terminal I/O stream as returned by the **Sysopen** function. To use the I2C extension bus use the **PCOM15.1**.

HMIBuiltInID (UDINT)

Management messages ID as returned bu the **HMIBuiltInMessages** FB.

FLed0~6 (BOOL)

Keys from [0] to [6] commands.

Backlight (BOOL)

Display backlight command.

Out0~1 (BOOL)

Logic output commands.

Enabled (BOOL)

Active if the FB is enabled.

Ready (BOOL)

Active if the Netlog terminal hardware is ready.

Fault (BOOL)

Active for a program loop on error.

OneWireOk (BOOL)	Active if at least one One-Wire device connected to the terminal connector.
OneWireTrig (BOOL)	Active for a program loop, it must be used as trigger to read OneWireData array.
OneWireData (BYTE[16])	Returns the data read from the device connected to the terminal connector.
FKey0~9 (BOOL)	Function keys from [0] to [9] status.
KeyFUN (BOOL)	FUN key status.
KeyESC (BOOL)	ESC key status.
KeyCLR (BOOL)	CLR key status.
KeyENT (BOOL)	ENT key status.
KeyUP (BOOL)	UP key status.
KeyDOWN (BOOL)	DOWN key status.
Inp0~1 (BOOL)	Digitalò inputs stayus.

Error codes

If an error occurs, the Fault output is activated and [**SysGetLastError**](#) can detect the error code.

- 10049010 **HMIBuildInID** not defined.
- 10049020 **HMIBuildInID** not correct.
- 10049100 FB not executed in background (Back) task.

Esempi

In this example a **Picoface** terminal is managed, the messages has been defined in the LogicLab program with the **MyHMI** name.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	HMI	HMIPicoface	Auto	No	FB Netlog terminal
2	MsgID	UINT	Auto	No	Displayed message ID

ST example

```
(* Here open the terminal communication port. *)

IF (SysFirstLoop) THEN
    HMI.File:=Sysfopen('PCOM15.1', 'rw'); (* File pointer *)
END_IF;

(* Here manages the Picoface terminal. *)

MsgID:=MyHMI.MsgID; (* Displayed message ID *)
HMI.HMIBuiltInID:=MyHMI.HMIBuiltInID; (* ID gestione messaggi *)
HMI(Enable:=TRUE);
```

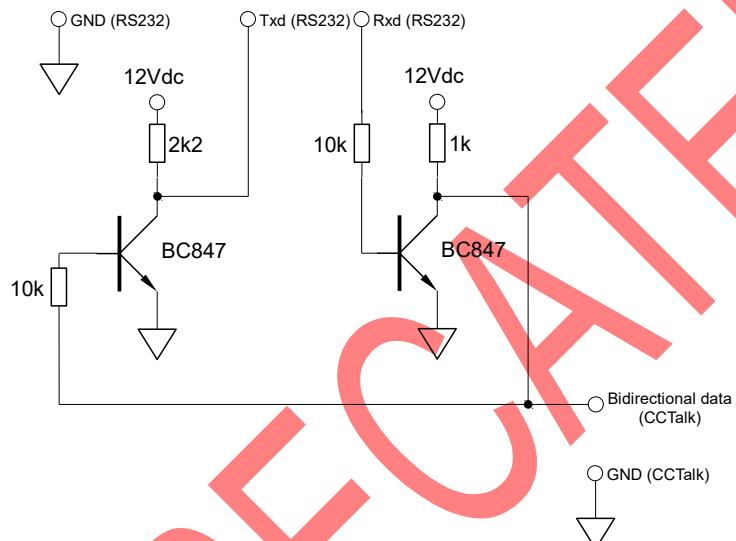
7.24 Cctalk protocol management library (eCCTalkProtoLib)

Note! To use the library it must be imported into your project. See the chapter [how to import libraries](#).

The ccTalk protocol is an open standard serial protocol designed to provide secure transfer of credit information and status for applications in the automated monetary transactions field. Peripherals such as the currency detectors for coins and banknotes found in a diverse range of automatic payment equipment such as transportation, ticketing, payphones, amusement machines, and retail cash management use ccTalk to talk to the host controller.

This library allows you to manage devices with ccTalk interface, a management protocol FB **ccTalkProtocol** connecting to a serial port allows to manage the protocol. Other Fbs linked to the **ccTalkProtocol** FB can manage equipments with ccTalk interface.

The ccTalk protocol operates on a single wire bus, all devices use an open collector configuration to connect in multi-drop to bus. To interface the cctalk bus with the RS232 serial port a special converter must be used.



7.24.1 ccTalkProtocol, manages ccTalk protocol

Type	Library
FB	eCCTalkProtoLib_A000

This function block manges the protocol, it must be connected to a serial port file to which you connect the cctalk devices. This function block is protected and require a code to unlock it (see [functions and function blocks protection](#)). It is possible to use it freely in test mode for 15 min.

The FB returns a **CCTalkProtOID** that must be passed to the linked FB for the management of the various devices connected ccTalk. The **SpyOn** if active allows you to spy on the FB operations.



The **Fault** output is activated for a program loop in case of error.

Enable (BOOL)	Protocol management enable.
SpyON (BOOL)	Active allows to spy the FB working.
File (FILEP)	Stream returned by Sysfopen function.
Enabled (BOOL)	Active on Enable command.
Fault (BOOL)	Active for a program loop if management error.
CCTalkProtOID (UDINT)	Protocol ID to pass to linked FBs.

Trigger di spy

If **SpyOn** is active the [SysSpyData](#) function is executed this allows to spy the FB operations. There are various levels of triggers.

TFlags Description

- 16#00000001 'Tx' ccTalk command sent.
- 16#00000002 'Rx' ccTalk answer received.

Error codes

If an error occurs, the **Fault** output is activated and [SysGetLastError](#) can detect the error code.

Code Description

- 10050010 **File** value not defined.
- 10050020 The FB is not executed on background (Back) task.
- 10050030 Protected FB. The available time for demo mode is over.
- 10050100 Communication timeout
- 10050200~4 Error in the sequences of sending command.
- 10050300~4 Error in the sequences of receiving command.

7.24.2 AlbericiAL66, Alberici AL66 coin acceptor

Type	Library
FB	eCCTalkProtoLib_A000

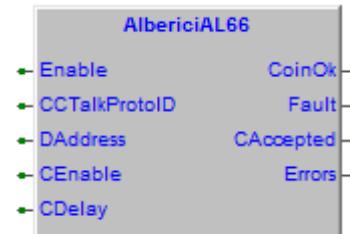
This function block manages a Alberici coin acceptor model AL66V, it must be linked with the **ccTalkProtocol** that manages the ccTalk protocol.

By activating the **Enable** input the dialogue with the coin acceptor with address defined in **DAddress**, starts. The FB communicates continuously with the acceptor sending commands every time defined in **CDelay**. I remember that to accept the coins, acceptor must be interrogated via cctalk at least every 500 mS.

The **CEnable** parameter allows you to define the currencies accepted by the acceptor, there is a bit pattern of 16 bits, where each bit corresponds to a type of currency (Refer to the acceptor documentation).

If the inserted coin is accepted the **CoinOk** is set for a program loop and on the **CAccepted** output (Bit pattern 16 bits) is set the bit relate to the currency accepted.

In case of error execution **Fault** is set for a program loop and Errors is incremented.



Enable (BOOL)	Enables the FB
CCTalkProtolD (UDINT)	Protocol management ID (From ccTalkProtocol.FB).
DAddress (USINT)	Device address, identifies the device to communicate with.
CEnable (WORD)	Coins enable, coins enable bit pattern.
CDelay (UINT)	Command delay, delay between commands (mS).
CoinOk (BOOL)	Active for a program loop if coin is been accpted.
Fault (BOOL)	Active for a program loop if management error.
CAccepted (WORD)	Coin accepted, coin accepted indicator, bit pattern.
Errors (UDINT)	Execution error counter.

Error codes

If an error occurs, the **Fault** output is activated and [**SysGetLastError**](#) can detect the error code.

- 10051010 **HMIBuildInID** not defined.
- 10051020 **HMIBuildInID** not correct.
- 10051030 The FB is not executed on background (Back) task.
- 10051040 Too many Fbs, not possible to execute.
- 10051100 Device communication error.

Example

In the example an Alberici AL66V electronic coin acceptor with ccTalk protocol is managed. The sample program defines the coins that can be accepted and counts the accepted coins according to their values.



Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Sm	SYSSERIALMODE	Auto	No	..		Serial mode
2	Fp	FILEP	Auto	No	..		File pointer
3	CCTalk	ccTalkProtocol	Auto	No	..		ccTalk protocol manager
4	Acpt	AlbericiAL66	Auto	No	..		Alberici acceptor
5	Credit	UDINT	Auto	No	..		Credit (Euro cents)
6	i	INT	Auto	No	..		Auxiliary variable
7	NrOfCoins	USINT	Auto	No	..		Number of inserted coins

ST example

```

(* ----- *)
(* INIZIALIZZAZIONI *)
(* ----- *)
(* Eseguo inizializzazioni. *)

IF (SysFirstLoop) THEN
    (* Eseguo apertura porta seriale. *)
    Fp:=Sysfopen('COM0', 'rw'); (* Port COM0 file pointer *)
    (* Eseguo configurazione porta seriale. *)
    i:=SysGetSerialMode(ADR(Sm), Fp);
    Sm.Baudrate:=9600; (* Baud rate *)
    Sm.Parity:='N'; (* Parity *)
    Sm.DataBits:=8; (* Data bits *)
    Sm.StopBits:=1; (* Stop bits *)
    Sm.DTRManagement:=DTR_AUTO_WO_TIMES; (* DTR management *)
    Sm.DTRComplement:=FALSE; (* DTR complement *)
    Sm.EchoFlush:=FALSE; (* Echo flush *)
    Sm.DTROffTime:=0; (* DTR off time *)
    Sm.DTROnTime:=0; (* DTR on time *)
    i:=SysSetSerialMode(ADR(Sm), Fp);

    (* Eseguo inizializzazione variabili. *)
    CCTalk.File:=Fp; (* File pointer *)
    Acpt.DAddress:=2; (* Device address *)
    Acpt.CDelay:=500; (* Command delay (mS) *)
END_IF;

(* ----- *)
(* GESTIONE PROTOCOLLO CCTALK *)
(* ----- *)
(* Gestione del protocollo ccTalk. *)

CCTalk(Enable:=TRUE); (* ccTalk protocol management *)

(* ----- *)
(* GESTIONE ACCETTATORE ALBERICI *)
(* ----- *)
(* Gestione accettatore Alberici. *)

Acpt.CCTalkProtoID:=CCTalk.CCTalkProtoID; (* Protocol ID *)

(* ----- *)
(* Definizione monete accettate da accettatore. Viene definita maschera *)
(* ----- *)

```

DEPRECATED

```
ut.CoinOK) THEN
Coins:=NrOfCoins+1; (* Numero di monete inserite *)

Alcolo denaro inserito nell'accettatore. *)

(Acpt.CAccepted) OF
0001: Credit:=Credit+200; (* 2 Euro *)
0002: Credit:=Credit+100; (* 1 Euro *)
0004: Credit:=Credit+50; (* 50 Centesimi *)
0008: Credit:=Credit+20; (* 20 Centesimi *)
0010: Credit:=Credit+10; (* 10 Centesimi *)
0020: Credit:=Credit+5; (* 5 Centesimi *)
0040: Credit:=Credit+2; (* 2 Centesimi *)
0080: Credit:=Credit+1; (* 1 Centesimo *)
CASE;
  of file] *)
```

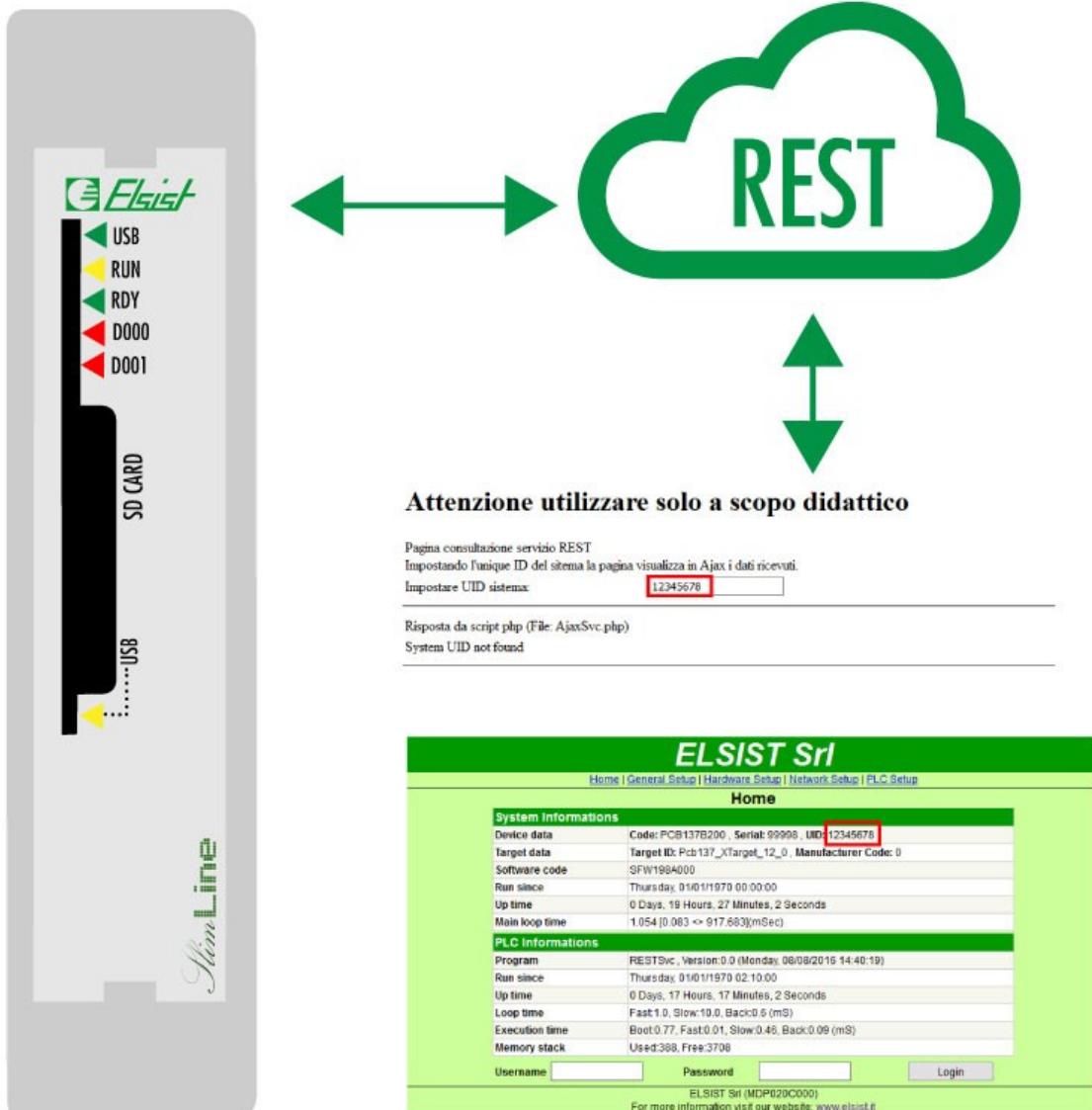
7.25 Library for manage REST Services (eLLabRESTSvcLib)

Warning ! For use this library it's necessary to import it in your project. See the chapter relative to library import.

REST is not an architecture or one standard, but it is a group of guidelines to create a system architecture. All of the communication between client and server takes place through HTTP then REST server can be hosted in a server farm. Exclusively for this project we have published one server on one free shared hosting : <http://slimline.altervista.org>.

Using the **RESTWSvcClient** FB the system send the generated event in the system, and without event the system, the system send ciclically the heartbeat frame to the cloud server and receive the reply with any server command. Using this client architecture all IP problem is bypassed, the system work also on NATted network because is the system that open the connection to the public IP of the cloud server.

Leaning on one FIFO the events to send are buffered locally with the relative timestamp, if there is connection the event is sent to the cloud server, but if there isn't connection the system try to send and send the event when the connection back on. With the event is send the relative timestamp and this allow the server to place the event in the time correctly.



7.25.1 RESTWSvcClient, connects to a REST web service

Type	Library
FB	eLLabRESTSvcLib_A220

This function block manage the connection to the REST server, for the HTTP protocol management is used the **HTTPGetPage** FB. To guarantee the data buffering for rapid event, the service use the FIFO stack on one file. In **FIFOFile** and **FIFOSize** you can choose which file to use for the FIFO stack and the stack dimension.

The management of the heartbeat message sending to the server, is provided at every time interval defined in **HbitTime**. This allow to control the right operation of the service in the absence of the message to send to the server. The output **RsvcOn** if is enable indicate that the service is on.

It is necessary provide the address of RESTSERVICBACKUP type in RESTSvcBck that will have to be allocated in a manner retentive way, because is not possible to manage retentive variables in a FB. And we have to manage the address of the FIFO stack by keeping them at the system stop.

Enable (BOOL)	Activation command to connect REST services.
SvcAck (BOOL)	Data receive acknowledge. It must be enable for one loop from user program on reading data from REST server.
SpyOn (BOOL)	If enabled allow to spy operation of this FB.
RPack (USINT)	Data number received from the server. Must be set from user program VÀ valorizzato da programma utente contemporaneamente at the same time of SvcAck with the number of parameter received from the server REST.
HBitTime (UINT)	Time interval for sending heartbeat to the server (S). Every defined interval, one heartbeat message is sent to the server.
FIFOFile (STRING[32])	Name and path of the support file for FIFO stack.
FIFOSize (UDINT)	Dimension (bytes) of the FIFO stack.
RESTSvcBck (@RESTSERVICEBACKUP)	Address allocation RESTSERVICEBACKUP structure. The structure it must be allocated in retentive memory.
Host (@USINT)	Server URL to connect to.
Port (UINT)	Port number of REST server to connect to. (Default 80).
BLength (UDINT)	Request buffer dimension and REST reply.
Page (@USINT)	Rest service manage page.
Enabled (BOOL)	Function block enable.
RSvcOn (BOOL)	REST service enable. It activates on communication with the enable REST server. It is deactivated if the communication drops, error on sending messages or heartbeat.
SvcOk (BOOL)	Server OK reception. It activates on server REST reply and remains enable until user program activates SvcAck . With the active bit , the data received from the server are available in the buffer pointed by Pbuffer .
Fault (BOOL)	Enable for one program loop if management error.
RPCCount (USINT)	Parameter numer received from the server.
RSVID (UDINT)	REST serive ID. REST service identifier, must be provided to the FB connected to service.
FSpace (UDINT)	Free byte in the support FIFO file. Indicate how much free space there is in the support FIFO buffer.
PBuffer (@USINT)	Pointer buffer received REST page. Allocated buffer in FB through RMAloc , it will come freeon the activation of SvcAck .
PktsOk (UDINT)	Counter of REST package correctly exchanged with the server. Reached max value the count resome from 0.
Resyncs (UDINT)	Resynchronization counter with the REST server. Reached max value, counter resumes from

0.

Messages exchanged between FB and server have an identifier that allows the control. In the case of misalignment (loss of a message) a resynchronization is performed.

Errors (UDINT)

Communication error counter with REST server. Reached max value, counter resumes from 0.

Error Code

In the case of error the Fault output activates, with [**SysGetLastError**](#) it is possible to read the error code.

Code Description

10057020 FB executed in different task than background task.

10057030 **RESTSvBck** is not defined.

10057050 Execution error FB **HTTPGetPage**.

10057100~7 Reading error from FIFO file.

10057110 Error reception message from FIFO file. FIFO is reinitialized.

10057200 FIFO message to send to the server more long of **BLength** (Message was deleted).

10057210 Heartbeat message to send to the server, more length of **BLength** (Message was not sent).

10057300 Error Errore ricezione risposta HTTP da server REST.

10057310 Message received from server does not have message identifier "MID".

10057311 Message received from server does not have parameter number "RP".

10057312 Message received from server does not have page data "Page".

7.25.2 RESTSendFct, sends a REST message

Type	Library
FB	eLLabRESTSvcLib_A220

This function block can be used as like as one function, allow to insert one message in the fifo buffer of REST client (FB **RESTWSvcClient**) connected. The connection between this FB and FB client happens through **RSvID** parameter to whom provide the output value corresponding to the FB **RESTSvcClient**.

In **Add** must be indicate the address of string to send, performing the FB the message it is inserted immediatly in the FIFO buffer fowarding and the REST client will send to the server as soon as possible.



Add (@USINT) Address of message string to send.

RSvID (UDINT) REST service ID. identifier of REST service, it must be passed to the FB connected to the service.

Fault (BOOL) Enable for one program loop if managment error.

Error Code

In the case of error the Fault output activates, with [SysGetLastError](#) il possible to read the error code.

Code	Description
10057020	FB executed in different task than background task.
10058030	RSvID is not defined.
10058040	RSvID incorrect.
10058100	The message to send is more length than BLength of FB RESTWSvcClient . The message is cut to the defined length.
10058110	There is no space in FIFO buffer to contain the message, the message is lost.
10058200~4	Writing error on FIFO file.

DEPRECATED

10058200~4 Writing error on FIFO file.

How to test REST services

The REST services managed by system provides a dialog with one cloud server over internet, then to explain the function we made a cloud on a free shared hosting <http://www.slimline.altervista.org>.

The architecture of the server is based on php scripts while the htm consultation page is supported by Jquery and use an ajax technique for the live update.

The cloud server is free to use for all. Just load on SlimLine the demo program PTP135A000 which is already configured to send data to cloud server.

Variables definition.

	Name	Type	Address	Array	Init value	Attribute	Description
1	DInp	USINT	Auto	[0..1]	..		Digital input byte
2	DOut	USINT	Auto	No	..		Digital output byte
3	i	INT	Auto	No	..		Auxiliary counter
4	LastError	UDINT	Auto	No	..		Last execution error number
5	REST	RESTWSvcClient	Auto	No	..		REST service FB
6	RESTSBf	STRING	Auto	[64]	..		REST Send buffer
7	RxPage	STRING	Auto	[64]	..		Received page (Only for debug)

ST Examples

```

(* ----- *)
(* ESEGUE INIZIALIZZAZIONI *)
(* ----- *)
(* Esegue inizializzazione variabili. *)

IF (SysFirstLoop) THEN
  REST.SpyOn:=TRUE;
  REST.RESTSvBck:=ADR(RESTSvBck); (* REST service backup pointer*)
  REST.FIFOFile:='Storage/REST.bin'; (* Path and name of file where to log *)
  REST.FIFOSize:=10000; (* FIFO file size *)

  REST.Host:=ADR('www.slimline.altervista.org'); (* Hostname servizio REST *)
  REST.Page:=ADR('/Mdp095a100/Ptp135a000/RESTSvc.php'); (* Pagina servizio REST *)
  REST.Port:=80; (* Porta servizio REST *)

  REST.HBitTime:=5; (* Heartbeat time (S) *)
  REST.BLength:=512; (* REST Request/Answer buffers length *)
END_IF;

(* ----- *)
(* GESTIONE SERVIZIO REST *)
(* ----- *)
(* Esegue gestione servizio REST. *)

REST(Enable:=TRUE); (* Esegue gestione servizio REST *)
RESTSend(RSvID:=REST.RSvID, Add:=NULL);
REST.SvcAck:=FALSE; (* REST service acknowledge *)
IF (REST.Fault) THEN LastError:=SysGetLastError(TRUE); END_IF;

(* ----- *)
(* INVIO INGRESSI DIGITALI AL SERVER CLOUD *)
(* ----- *)
(* Esegue compattazione ingressi su byte. *)

DInp[0]:=16#00; (* Digital input byte *)
IF (Di00CPU) THEN DInp[0]:=DInp[0] OR 16#01; END_IF;
IF (Di01CPU) THEN DInp[0]:=DInp[0] OR 16#02; END_IF;
IF (Di02CPU) THEN DInp[0]:=DInp[0] OR 16#04; END_IF;
IF (Di03CPU) THEN DInp[0]:=DInp[0] OR 16#08; END_IF;
IF (Di04CPU) THEN DInp[0]:=DInp[0] OR 16#10; END_IF;
IF (Di05CPU) THEN DInp[0]:=DInp[0] OR 16#20; END_IF;

(* Il FB "RESTWSvcClient" invia un messaggio di heartbeat al server ogni *)
(* tempo impostato in "HBitTime". Per informare il server dello stato *)
(* degli ingressi digitali lo invio su ogni variazione. *)

IF (DInp[0] <> DInp[1]) THEN
  DInp[1]:=DInp[0]; (* Digital input byte *)
  i:=SysVarsprintf(ADR(RESTSBf), SIZEOF(RESTSBf), 'DInp=%d', USINT_TYPE, ADR(DInp[0]));
  RESTSend(Add:=ADR(RESTSBf));

```

```

END_IF;

(* ----- *)
(* RICEZIONE USCITE DIGITALI DA SERVER CLOUD *)
(* ----- *)
(* Su ricezione Ok da servizio REST controllo l'Ok ricevuto. *)

IF NOT(REST.SvcOk) THEN RETURN; END_IF;
REST.SvcAck:=TRUE; (* REST service acknowledge *)

(* Trasferisco pagina ricevuta in buffer. Solo per debug permette di *)
(* visualizzare il contenuto della pagina ricevuta. *)

i:=TO_INT(Sysmemmove(ADR(RxPage), REST.PBuffer, SIZEOF(RxPage)));

(* Eseguo acquisizione valore uscite ricevuto da server. *)

REST.RPack:=0; (* REST parameters acknowledge *)
IF (SysVarsscanf(SysStrFind(REST.PBuffer, ADR('DOut='), FIND_GET_END), '%d', USINT_TYPE, ADR(DOut))) THEN
REST.RPack:=REST.RPack+1; END_IF;

(* Eseguo gestione uscite digitali. *)

Do00CPU:=TO_BOOL(DOut AND 16#01); //Do00 CPU module
Do01CPU:=TO_BOOL(DOut AND 16#02); //Do01 CPU module
Do02CPU:=TO_BOOL(DOut AND 16#04); //Do02 CPU module
Do03CPU:=TO_BOOL(DOut AND 16#08); //Do03 CPU module

(* [End of file] *)

```

Come si vede viene parametrizzato il FB **REST** con i parametri per la connessione al nostro server cloud di prova.

```

REST.Host:=ADR('www.slimline.altervista.org'); (* Hostname servizio REST *)
REST.Page:=ADR('/Mdp095a100/Ptp135a000/RESTSvc.php'); (* Pagina servizio REST *)
REST.Port:=80; (* Porta servizio REST *)

```

Il FB **REST** invia un messaggio al server cloud ogni tempo definito in **HbitTime**. Su variazione stato ingressi digitali il FB **RESTSend** comanda l'invio al server cloud dello stato degli ingressi.

Ad ogni ricezione di un messaggio REST il server cloud registra i dati ricevuti in un file ini di appoggio ed invia al sistema il valore di comando delle uscite. Riporto di seguito listato del file **RESTSvc.php** a cui si connette il FB REST anche se il file sorgente si trova nel programma dimostrativo.

```

<?php
// ****
// FUNZIONI CONVERSIONE DATI RICEVUTI
// ****
// Funzioni per conversione dati.

function RxBYTE($Rx, $Ofs) {return intval(substr($Rx, $Ofs, 2), 16);}
function RxWORD($Rx, $Ofs) {return intval(substr($Rx, $Ofs, 4), 16);}
function RxDWORD($Rx, $Ofs) {return intval(substr($Rx, $Ofs, 8), 16);}
function RxREAL($Rx, $Ofs) {$Pk=pack("L", intval(substr($Rx, $Ofs, 8), 16)); $Uk=unpack("f", $Pk);
return ($Uk[1]);}

// -----
// INCLUSIONE FILES
// -----
// Inclusione files.

$HomeDir=substr($_SERVER['SCRIPT_FILENAME'],0,-strlen($_SERVER['SCRIPT_NAME'])); //Home directory
require_once($HomeDir."/Mdp095a100/Ptp135a000/Include.php"); //Inclusioni generali

// -----
// CONTROLLO RICHIESTA IN ARRIVO
// -----
// La richiesta deve contenere i campi, MID, UID, MV, RP. Se errore esco.

if (!CkReqPars(array("MID", "UID", "MV", "RP"))) exit("Wrong REST parameters");
if (!is_numeric($_REQUEST['UID'])) exit("Wrong system UID");

// Per ogni sistema (Riconoscibile dal suo "UID") esiste un file dati ne eseguo
// lettura e compilazione array globale.

ReadINIFile($_REQUEST['UID']); //File dati di sistema
$GLOBALS['Dt']['PollTime']=GetuTime()-$GLOBALS['Dt']['Heartbeat']; //Tempo poll sistema

```

```

$GLOBALS['Dt']['Heartbeat']=GetuTime(); //Data/Ora ultimo heartbeat (UTC)
// Salvo messaggio ricevuto.

$GLOBALS['Dt']['MV']=$_REQUEST['MV']; //Message version
$GLOBALS['Dt']['RP']=$_REQUEST['RP']; //Received parameters
$GLOBALS['Dt']['RxMessage']="MID={$GLOBALS['Dt']['MID']}, UID={$REQUEST['UID']}, MV={$GLOBALS['Dt']['MV']}, RP={$GLOBALS['Dt']['RP']}"; //Last request
if (isset($_REQUEST['Data'])) $GLOBALS['Dt']['RxMessage'].=", Data={$REQUEST['Data']}";

// -----
// CONTROLLO ID MESSAGGIO
// -----
// Controllo se ricevuto l'acknowledge dallo SlimLine del messaggio REST
// inviato precedentemente dal server. Controllo se il MID ricevuto è
// corretto (Successivo al MID del messaggio precedente).

if ((($_REQUEST['MID']-$GLOBALS['Dt']['MID'])&0xFFFF) == 1)
{
    // Ricevuto MID successivo messaggio corretto (Nessun messaggio è
    // andato perso) utilizzo MID ricevuto.

    $GLOBALS['Dt']['MID']=$_REQUEST['MID']; //Message ID
}
else
{
    // Errore ricezione messaggi, occorre eseguire una resincronizzazione
    // sistema, viene inviato un numero random che sarà utilizzato dal
    // sistema come prossimo MID.

    $GLOBALS['Dt']['MID']=rand(0, 65535); //Message ID
    $GLOBALS['Dt']['Resyncs']++; //REST resyncronizations
}

// -----
// ACQUISIZIONE INFORMAZIONI DAL MESSAGGIO DATI
// -----
// Un messaggio dati contiene un campo "Data" composto da diversi campi, ogni
// byte occupa due caratteri ascii. I dati sono in Big endian, MSB ... LSB.
// +-----+-----+
// | Length|0|0| Epoch | Value |
// +-----+-----+-----+-----+
// Length: Lunghezza record (2 byte)
// Epoch: Epoch time (4 byte)
// Value: Stringa con valore (Lunghezza variabile)
// -----
// Se messaggio ricevuto contiene campo "Data" eseguo acquisizione dati campo.

if (!isset($_REQUEST['Data'])) goto SENDDATA;
$GLOBALS['Dt']['Length']=RxWORD($_REQUEST['Data'], 0); //Lunghezza record dati
$GLOBALS['Dt']['Epoch']=RxDWORD($_REQUEST['Data'], 8); //Epoch time relativo al record dati
$GLOBALS['Dt']['Value']=substr($_REQUEST['Data'], 16, ($GLOBALS['Dt']['Length']-8)); //Stringa valore record dati

// Nel campo "Value" il sistema SlimLine invia le variabili indicandole nel modo
// "Var1=Valore|Var2=Valore|..." controllo che vi sia almeno un "=".
// Nel nostro esempio vi sarà una sola variabile "DInp=xx".

if (strpos($GLOBALS['Dt']['Value'], "=") === false) goto SENDDATA;

// Estraggo nome e valore delle variabili creando un array associativo.
// Mi troverò con $DtArray('DInp' => xx, ...)

$DtArray=array(); //Array associativo dati ricevuti
$Variables=explode("|", $GLOBALS['Dt']['Value']); //Variables array
foreach ($Variables as &$Variable)
{
    if (strpos($Variable, "=") !== false)
    {
        $VNameValue=explode("=", $Variable); //Array Nome/Valore variabile
        $DtArray=array_merge($DtArray, array($VNameValue[0] => $VNameValue[1]));
    }
}

// Appoggio stato ingressi digitali.

$GLOBALS['Dt']['DInp']=$DtArray['DInp']; //Stato ingressi digitali

```

```
// -----
// INVIO DATI AL SISTEMA
// -----
// Eseguo scrittura file per storicizzare i dati sul server.

SENDATA:
WriteINIFile($_REQUEST['UID']); //Scrittura file ini

// Inserisco la definizione dei campi da impostare, separo ogni campo con
// lo spazio per permettere nel sistema alla scanf di interrompersi sulla
// acquisizione di valori stringa. Nel nostro esempio vi è un solo campo.

$RetPars=sprintf("DOut=%d ", $GLOBALS['Dt']['DOut']);

// -----
// RITORNO PAGINA AL CLIENT
// -----
// Compilo messaggio di risposta che inizia con il MID. Il valore ritornato
// è calcolato sommando il valore di UID. In questo modo si garantisce che
// il sistema che riceve il messaggio possa verificalo utilizzando il suo
// unique ID.

$GLOBALS['Dt']['TxMessage']=sprintf("MID=%d", ($GLOBALS['Dt']['MID']+$_REQUEST['UID'])&0xFFFF); //Return page
$GLOBALS['Dt']['TxMessage'].=sprintf("&RP=%d", 0); //Return page
$GLOBALS['Dt']['TxMessage'].=sprintf("&Page=%s", $RetPars); //Return page
echo $GLOBALS['Dt']['TxMessage'];
?>
```

As shown in the program code we have realized a more simple management based on one .ini file, but who have familiarity whit web applications ma in realtà chi ha dimestichezza con applicazioni web troverà will find more efficient to use one database.

DEPRECATED

How service work

As we have seen, the SlimLine system send data to the cloud server (Execute RESTSvc.php script sending data through GET) that based on unique ID, the system detect if there already one file ini dedicated to the system on the server. If exist the server execute the reading, if not, this file is created, in this file are stored all data neccessary to manage the service. Here is an example of the file ini:

```
MID="19867"
MV="1.0"
RP="1"
Length="14"
Epoch="1470474898"
Value="DIInp=2"
DIInp="2"
DOut="1"
RxMessage="MID=19866, UID=10978974, MV=1.0, RP=1"
TxMessage=
Resyncs="1"
PollTime="5.3649678230286"
Heartbeat="1470474724.041"
```

As seen, is stored the MID, that is the progressive ID of the message which allows to control the incoming data. All other fields are stored for debug purpose, to understand the function of the service.

Data are updated every message reception from the system (Heartbeat time or input variation) and in the field **PollTime** the php script calculate the time between updates.

For data displaying, see the htm page at the link <http://www.slimline.altervista.org>. We have something like this:

In the page there is one alert that tell to use this service for only didactic purpose because there is no password or other protection. It is evident that by inserting one random UID of the system is possible to find the UID of our system and control the output.

All of this was done deliberately to limit the complexity of the project, if you want to use this project is possible to add password and more function by starting developing from the source code.

UniqueId (UID) of the system can be displayed in debug from LogicLab but is also visible in the webpage.

The screenshot shows a web browser window with the URL <http://192.168.0.183/System/Home.htm>. The page title is "SlimLine - Home". The main content area has a green header with the text "ELSIST Srl". Below the header is a navigation bar with links: Home | General Setup | Hardware Setup | Network Setup | PLC Setup. The main content is titled "Home" and contains a section titled "System Informations". It displays three rows of data:

Device data	Code: MPS046B200 , Serial: 1 , UID: 2097153
Target data	Target ID: Mps046_XTarget_12_0 , Manufacturer Code: 0
Software code	SFW184B030

The htm page use Jquery and is live updated with ajax call. We do not explain the tags because are obvious. almost entirely the page is based on div element that is updated by javascript. Here is the source code.

```

// -----
// FUNZIONE ESEGUITA SU LOAD PAGINA
// -----
// Sul load della pagina attivo ajax.

$(document).ready(function()
{
    AjaxCall(); //Eseguo chiamata ajax su load pagina
    setInterval(AjaxCall, 5000); //Imposto chiamata ciclica ajax
});

// -----
// RICHIESTA AJAX
// -----
// Viene eseguita la richiesta ajax. Eseguo lo script "AjaxSvc.php" passando
// i parametri in POST.

function AjaxCall()
{
    // Compongo byte di gestione comando output.

    var DOut=0x00; //Digital output byte command
    if ($("#Do00CPU").is(':checked')) DOut+=0x01;
    if ($("#Do01CPU").is(':checked')) DOut+=0x02;
    if ($("#Do02CPU").is(':checked')) DOut+=0x04;
    if ($("#Do03CPU").is(':checked')) DOut+=0x08;

    // Eseguo invio richiesta ajax con parametri in POST.

    $.ajax(
    {
        type:"POST",
        url:"/Mdp095a100/Ptp135a000/AjaxSvc.php",
        data:"UID="+$("#UID").val()+"&DOut="+DOut,
        dataType:"html",

        // Funzione eseguita su successo della chiamata.

        success:function(Answer)
        {
            // Copio stringa ricevuta da script php per visualizzazione.

            $("div#Answer").html(urldecode(Answer));

            // Suddivido campi, sono separati dal carattere "|".

            var AArray=Answer.split('|'); //Answer array
            for(var i=0; i<AArray.length; i++)
            {
                // Suddivo i campi tra nome campo e valore. Valorizzo il
                // div di visualizzazione valore.

                var VArray=AArray[i].split('=');
                $("div#+VArray[0]").html(urldecode(VArray[1]));
            }

            // Eseguo valorizzazione radio button stato ingressi digitali.
        }
    }
}

```

```
$("#Di00CPU").prop("checked", ($("#div#DInp").html() &0x01?true:false));
$("#Di01CPU").prop("checked", ($("#div#DInp").html() &0x02?true:false));
$("#Di02CPU").prop("checked", ($("#div#DInp").html() &0x04?true:false));
$("#Di03CPU").prop("checked", ($("#div#DInp").html() &0x08?true:false));
$("#Di04CPU").prop("checked", ($("#div#DInp").html() &0x10?true:false));
$("#Di05CPU").prop("checked", ($("#div#DInp").html() &0x20?true:false));
},
// Funzione eseguita su errore chiamata.

error: function()
{
  $("#div#Answer").html("Call error");
}
});
}

// -----
// DECODIFICA CARATTERI URL
// -----
// Per evitare che nel campo valore vi siano caratteri "=" che possono dare
// fastidio nello split converto campo dati in entità URL e con questa
// funzione ne eseguo la decodifica.

function urldecode (str) {return decodeURIComponent((str + '').replace(/\+/g, '%20'));}
</script>
```

It is interesting to note that the to avoid character “=” in middle of the string, these are coded as URL request by the php script and than must be decoded from javascript.

DEPRECATED

The ajax request execute on the server the script **AjaxSvc.php** here are the source code.

```

// -----
// INCLUSIONE FILES
// -----
// Inclusione files.

$HomeDir=substr($_SERVER['SCRIPT_FILENAME'],0,-strlen($_SERVER['SCRIPT_NAME'])); //Rilevo Home directory
require_once($HomeDir."/Mdp095a100/Ptp135a000/Include.php"); //Inclusioni generali

// -----
// CONTROLLO RICHIESTA IN ARRIVO
// -----
// La richiesta deve contenere i campi, UID, DOut. Se errore esco.

if (!CkReqPars(array("UID", "DOut"))) exit("Wrong REST parameters");
if (!is_numeric($_REQUEST['UID'])) exit("Wrong system UID");

// -----
// ESEGUE LETTURA FILE DATI DI SISTEMA
// -----
// Per ogni sistema (Riconoscibile dal suo "UID") esiste un file dati ne eseguo
// lettura e compilazione array globale.

if (!ReadINIFile($_REQUEST['UID'])) exit("System UID not found");

// Salvo byte comando uscite digitali.

$GLOBALS['Dt']['DOut']=$_REQUEST['DOut']; //Comando uscite digitali

// Esegue scrittura file.

WriteINIFile($_REQUEST['UID']);

// -----
// RITORNO DATI A PAGINA WEB
// -----
// Ritorno dati a pagina web. Per evitare che nel campo valore vi siano
// caratteri "=" che danno fastidio nello split converto stringhe in entità URL.

$Return="MID={$GLOBALS['Dt']['MID']}"; //Message ID
$Return.="|MV={$GLOBALS['Dt']['MV']}"; //Message version
$Return.="|RP={$GLOBALS['Dt']['RP']}"; //Received parameters
$Return.="|Length={$GLOBALS['Dt']['Length']}"; //Lunghezza record dati
$Return.="|Epoch={$GLOBALS['Dt']['Epoch']}"; //Epoch time relativo al record dati
$Return.="|Value=".urlencode($GLOBALS['Dt']['Value']); //Stringa valore record dati
$Return.="|DInp={$GLOBALS['Dt']['DInp']}"; //Stato ingressi digitali
$Return.="|DOut={$GLOBALS['Dt']['DOut']}"; //Comando uscite digitali
$Return.="|RxMessage=".urlencode($GLOBALS['Dt']['RxMessage']); //Messaggio ricevuto
$Return.="|TxMessage=".urlencode($GLOBALS['Dt']['TxMessage']); //Messaggio trasmesso
$Return.="|Resyncs={$GLOBALS['Dt']['Resyncs']}"; //REST resyncronizations
$Return.="|PollTime={$GLOBALS['Dt']['PollTime']}"; //Tempo poll sistema
$Return.="|Heartbeat={$GLOBALS['Dt']['Heartbeat']}"; //Data/Ora ultimo heartbeat (UTC)
echo $Return;

```

How to test the demo

For test you just load the program on one SlimLine system connected to the business network and suitably configured to access the internet (Set Gateway and DNS server). The program is preconfigured with the URL of the Altevista services and with the address of the script to execute. As you can see in the source, the spy of the **RESTWSvcClient** FB is enable than the SpyConsole is possible to verify all the package from the system and the cloud server. Below the screenshot of Toolly with the activation of the total spy console **SpyData<CR>** (All package are bugged).

```

Toolly by ELSIST Srl - [Terminal]

File Utilities Devices Window Help

Elsist SlimLine (SFW198A000), maintenance shell
Code: PCB137B200, Serial:99998
Login: Admin
Password: *****

[Admin]> SpyData
Spy data active, type "Ctrl-C" to exit...
11:52:05(10.4) |Tx|GET //RESTServer/RESTSvc.php?
11:52:05(.001) |Rq|MID=22816&UID=10978974&MV=1.0&RP=1
11:52:05(.012) |Tx|Host: www.slimline.altervista.org..
11:52:05(.012) |Tx|Accept-Language: it-IT..
11:52:05(.186) |Tx|Accept: text/html, */*.
11:52:06(.211) |Tx|User-Agent: Elsist HTTPGetPage/SFR079A000..
11:52:06(.212) |Tx|Connection: Keep-Alive..
11:52:06(.409) |Rx|HTTP/1.1 200 OK..
11:52:06(.038) |Rx|Date: Mon, 08 Aug 2016 09:52:56 GMT..
11:52:07(.197) |Rx|Server: Apache..
11:52:07(.216) |Rx|Vary: Accept-Encoding..
11:52:07(.209) |Rx|Content-Length: 27..
11:52:07(.219) |Rx|Connection: Keep-Alive: timeout=1, max=100..
11:52:07(.206) |Rx|Content-Type: text/html..
11:52:08(.212) |Rx|Content-Type: text/html..
11:52:08(.197) |Rx|..
11:52:08(.211) |Rc|MID=57278&RP=0&Page=DOut=3
11:52:13(5.22) |Tx|GET //RESTServer/RESTSvc.php?
11:52:13(.001) |Rq|MID=22817&UID=10978974&MV=1.0&RP=1
11:52:13(.011) |Tx|Host: www.slimline.altervista.org..

```

The **Tx** sent and the **Rx** received string, have next to it the time interval from the next execution. The data send are labeled with **Rq** and **Rc** the data received.

By activating the trigger on the spycommand **SpyData -t 0x0000000C<CR>**, we will see only the data sent and received without visualize all the HTTP manage string.

```

Toolly by ELSIST Srl - [Terminal]

File Utilities Devices Window Help

Elsist SlimLine (SFW198A000), maintenance shell
Code: PCB137B200, Serial:99998
Login: Admin
Password: *****

[Admin]> SpyData -t 0x0000000C
Spy data active, type "Ctrl-C" to exit...
11:54:18(10.4) |Rq|MID=22833&UID=10978974&MV=1.0&RP=1
11:54:18(.361) |Rc|MID=57295&RP=0&Page=DOut=3
11:54:23(5.01) |Rq|MID=22834&UID=10978974&MV=1.0&RP=1
11:54:24(.449) |Rc|MID=57296&RP=0&Page=DOut=3
11:54:24(.516) |Rq|MID=22835&UID=10978974&MV=1.0&RP=1&Data=000E000057A856D0DInp=1
11:54:25(.338) |Rc|MID=57297&RP=0&Page=DOut=3
11:54:26(1.04) |Rq|MID=22836&UID=10978974&MV=1.0&RP=1&Data=000E000057A856D2DInp=0
11:54:26(.500) |Rc|MID=57298&RP=0&Page=DOut=3
11:54:31(5.01) |Rq|MID=22837&UID=10978974&MV=1.0&RP=1
11:54:32(.549) |Rc|MID=57299&RP=0&Page=DOut=3
11:54:37(5.01) |Rq|MID=22838&UID=10978974&MV=1.0&RP=1
11:54:37(.370) |Rc|MID=57300&RP=0&Page=DOut=3
11:54:42(5.01) |Rq|MID=22839&UID=10978974&MV=1.0&RP=1
11:54:43(.511) |Rc|MID=57301&RP=0&Page=DOut=3

```

From the analysis we will see that normally every 5 second one heartbeat package was send, but when there is a change in one digital input, one message with **Data** field that contain the state of the input was immediatly send.

DEPRECATED

8 Communication protocols

8.1 Modbus protocol

Modbus is a serial communication protocol became a de facto standard in industrial communication, and is now the most widely used connection protocol among industrial electronics devices. It is a protocol based on request/response and offers services specified by function codes.

SlimLine supports the Modbus RTU protocol on the serial ports and Modbus over IP with Ethernet connection on port **502**. The Modbus RTU protocol on the serial port has the default communication parameters **115200, e, 8, 1** and the node address for both serial port and TCP/IP is **1**.

8.1.1 Access to variables from modbus

The modbus functions allow to access to **MX100** user memory. The supported functions are:

Code	Function	Object type	Access	Address range
01h	Read coil status	Array of bits	Read	40000-44095 (20000-24095) (Note 1)
02h	Read input status	Array of bits	Read	40000-44095 (20000-24095) (Note 1)
03h	Read holding registers	Array of words (16 Bit)	Read	40000-42047 (20000-22047) (Note 2)
04h	Read input registers	Array of words (16 Bit)	Read	40000-42047 (20000-22047) (Note 2)
05h	Force single coil	Single bit	Write	40000-44095 (20000-24095) (Note 1)
06h	Preset single register	Single word (16 Bit)	Write	40000-42047 (20000-22047) (Note 2)
08h	Loopback diagnostic test			
0Fh	Force multiple coils	Array of bits	Write	40000-42047 (20000-22047) (Note 1)
10h	Preset multiple registers	Array of words (16 Bit)	Write	40000-42047 (20000-22047) (Note 2)

From software version SFW167D000 the addressable area is also in the range from 20000 to 2xxxx.

Note 1) In the functions that access the single-bit (every bit equals to one byte of memory), the address of the variable in the command is used. So having to access to the **MX100.50** location, the address value will be **40050**.

Note 2) In the functions that access the registers (16 bits) the address of the variable divided by 2 is used. So having to reach the **MX100.50** location, the value **40025** will be used.

8.1.2 Reading variables from modbus

To read variables, the **Read Holding registers** (code **0x03**) command is used. Assuming you have to read a **DWORD** variable allocated in memory at **MX100.64** address, this is the formula to calculate the address:

$$((\text{Address of variable}/2)+\text{Offset})-1 \rightarrow ((64/2)+40000)-1=40031 \rightarrow 0x9C5F$$

Being a **DWORD** variable, we read 2 consecutive registers starting from address allocation. Assuming that the value of the variable is **0x12345678**, we have:

Modbus RTU frames

Command frame: **01 03 9C 5F 00 02 DA 49**

Answer frame: **01 03 04 56 78 12 34 66 D5**

Modbus TCP/IP frames

Command frame: **00 00 00 00 00 06 01 03 9C 5F 00 02**

Answer frame: **00 00 00 00 00 07 01 03 04 56 78 12 34**

The representation of data in SlimLine is **Little-Endian**. The numbering starts from the least significant byte and ending with the most significant. So as you can see from the response string, the value of the 32-bit variable **0x12345678** is returned in two 16-bits registers with values **0x5678, 0x1234**.

8.1.3 Writing variables from modbus

To write variables, the **Preset multiple registers** (code **0x10**) command is used. Assuming you have to write to a **DWORD** variable allocated in memory at **MX100.64** address, this is the formula to calculate the address:

$$((\text{Address of variable}/2)+\text{Offset})-1 \rightarrow ((64/2)+40000)-1=40031 \rightarrow 0x9C5F$$

Being a **DWORD** variable, we will write 2 consecutive registers starting from address allocation. Assuming that we need to write the value **0x12345678** in the variable, we have:

Modbus RTU frames

Command frame: **01 10 9C 5F 00 02 04 56 78 12 34 D3 33**

Answer frame: **01 10 9C 5F 00 02 5F 8A**

Modbus TCP/IP frames

Command frame: **00 00 00 00 00 0B 01 10 9C 5F 00 02 04 56 78 12 34**

Answer frame: **00 00 00 00 00 06 01 10 9C 5F 00 02**

The representation of data in SlimLine is **Little-Endian**. The numbering starts from the least significant byte and ending with the most significant. So as you can see from the response string, the 32-bit value to write **0x12345678** is splitted in two 16-bits registers with values **0x5678**, **0x1234**.

8.1.4 Access to Real time clock from modbus

It is possible to access to real time clock data using Modbus commands to access registers. The supported functions are:

Code	Function	Tipo oggetto	Accesso	Range indirizzo
03h	Read holding registers	Array of words (16 Bit)	Read	100-105 (150 for Epoch time)
04h	Read input registers	Array of words (16 Bit)	Read	100-105 (150 for Epoch time)
06h	Preset single register	Array of words (16 Bit)	Write	100-105 (150 for Epoch time)
10h	Preset multiple registers	Array of words (16 Bit)	Write	100-105 (150 for Epoch time)

The registers (16 bits) of the real time clock are allocated in consecutive locations starting from the Modbus address 100. The registers contain the current value of the real time clock and writing a new value, the the real time clock will be automatically updated.

Address	Register	Note
100	Second	Second value (Range from 0 to 59)
101	Minute	Minute value (Range from 0 to 59)
102	Hour	Hour value (Range from 0 to 23)
103	Day	Day value (Range from 1 to 31)
104	Month	Month value (Range from 1 to 12)
105	Year	Year value (Range from 1900 to 2037)

8.1.5 Reading RTC from modbus

To read the values of the real time clock, the **Read Holding registers** (code **0x03**) command is used. We have to read 6 consecutive registers starting from the allocation address. The Modbus addressing requires an offset of 1, so **99 (0x0063)**.

Modbus RTU frames

Command frame: **01 03 00 63 00 06 35 D6**

Answer frame: **01 03 0C 00 1E 00 30 00 0B 00 1D 00 09 07 DA A2 32**

Modbus TCP/IP frames

Command frame: **00 00 00 00 00 06 01 03 00 63 00 06**

Answer frame: **00 00 00 00 00 0F 01 03 0C 00 1E 00 30 00 0B 00 1D 00 09 07 DA**

As you can see from the answer, the RTC values is:

Second: 30 (**0x001E**)

Minute: 48 (**0x0030**)

Hour: 11 (**0x000B**)

Day: 29 (**0x001D**)

Month: 9 (**0x0009**)

Year: 2010 (**0x07DA**)

8.1.6 Writing RTC from modbus

To write the values of the real time clock, the **Preset multiple registers** (code **0x10**) command is used. We have to write 6 consecutive registers starting from the allocation address. The Modbus addressing requires an offset of 1, so **99 (0x0063)**. Assume that we have to set these real time clock values:

Second: 30 (**0x001E**)

Minute: 48 (**0x0030**)

Hour: 11 (**0x000B**)

Day: 29 (**0x001D**)

Month: 9 (**0x0009**)

Year: 2010 (**0x07DA**)

Modbus RTU frames

Command frame: **01 10 00 63 00 06 08 00 1E 00 30 00 0B 00 1D 00 09 07 DA 5D C8**

Answer frame: **01 10 00 63 00 06 B0 15**

Modbus TCP/IP frames

Command frame: **00 00 00 00 00 13 01 10 00 63 00 06 08 00 1E 00 30 00 0B 00 1D 00 09 07 DA**

Answer frame: **00 00 00 00 00 06 01 10 00 63 00 06**

DEPRECATED

8.1.7 Epoch time access from modbus

There is also a 32 bits value for date/time Epoch time. The access to this read/write register is always performed using two 16 bits registers.

Address	Register	Note
150	Epoch time	Epoch time

8.1.8 Reading Epoch time from modbus

To read epoch time, the **Read Holding registers** (code **0x03**) command is used. We read 2 consecutive registers starting from the allocation address. The Modbus addressing requires an offset of 1, so **149 (0x0095)**.

Modbus RTU frames

Command frame: **01 03 00 95 00 02 D4 27**

Answer frame: **01 03 04 30 B5 4C A3 90 6C**

Modbus TCP/IP frames

Command frame: **00 00 00 00 00 06 01 03 00 95 00 02**

Answer frame: **00 00 00 00 00 07 01 03 04 30 B5 4C A3**

As you can see from the answer, the value is: **0x4CA330B5 → 1285763253 → GMT: Wed, 29 Sep 2010 12:27:33 UTC.**

8.1.9 Writing Epoch time from modbus

To write the epoch time value, the **Preset multiple registers** (code **0x10**) command is used. We have to write 2 consecutive registers starting from the allocation address. The Modbus addressing requires an offset of 1, so **149 (0x0095)**. Assume that we have to set these value:

GMT: Wed, 29 Sep 2010 12:27:33 UTC → 1285763253 → 0x4CA330B5

Modbus RTU frames

Command frame: **01 10 00 95 00 02 04 30 B5 4C A3 50 A3**

Answer frame: **01 10 00 95 00 02 51 E4**

Modbus TCP/IP frames

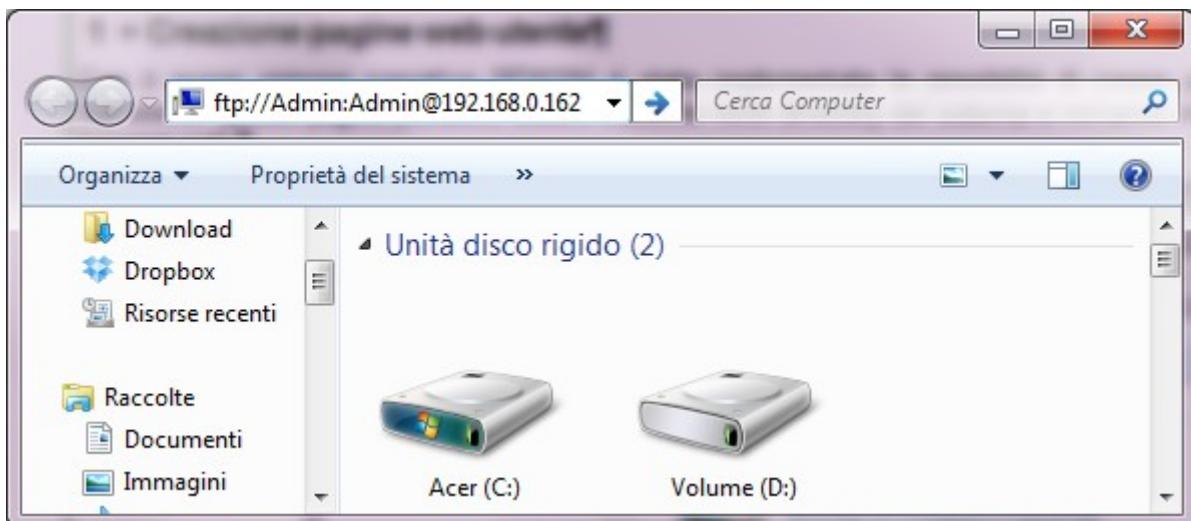
Command frame: **00 00 00 00 00 0B 01 10 00 95 00 02 04 30 B5 4C A3**

Answer frame: **00 00 00 00 00 06 01 10 00 95 00 02**

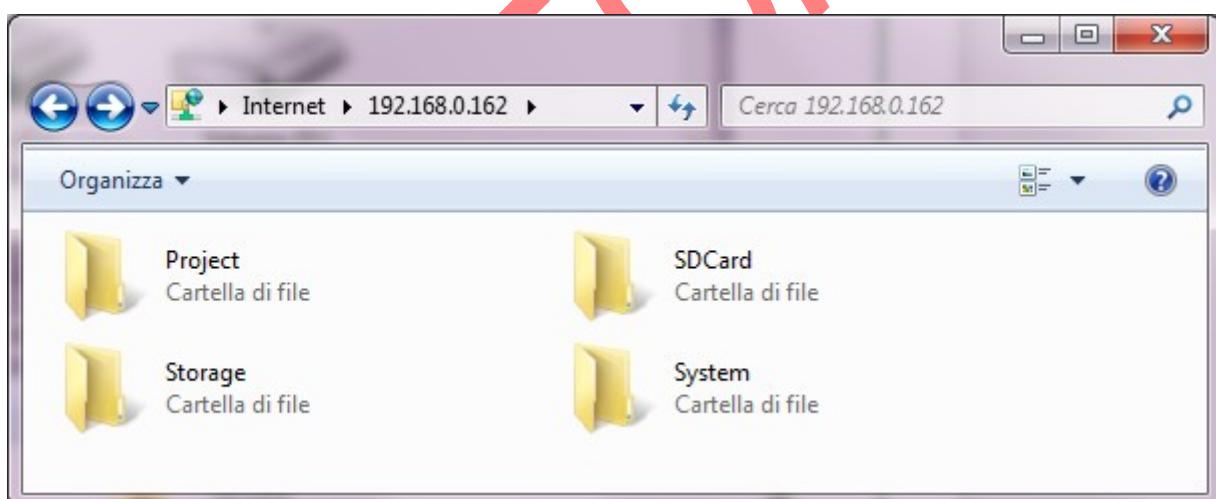
9 How to create user web pages

With the new SFW184 operating system has been implemented the possibility to create web pages directly by the user, these pages can be transferred in the system directory and will be displayed and accessed from web browsers.

To transfer the web pages created by the user in the SlimLine file system system, must be used an FTP client (Example FileZilla) but you can also simple use the Windows Explorer. as shown in the figure below, setting the credentials in the address bar and the IP address **ftp://Admin:Admin@192.168.0.162**, you can connect and see the file system.



Here's the view of the file system at the connection. **Project** and **System** folders are reserved for system use and you should not change its contents. The files of the user pages can be transferred in the **Storage** and **SDCard** (if present) folders.



So the user can create his own web pages using any HTML editor but also simply by using a simple text editor such as Notepad, of course it's needed to know the HTML syntax. The created pages will be transferred in the desired directory and accessing to them by a browser the page will be displayed.

9.1 Web pages criteria

Of course, the SlimLine web server has only a limited set of functions, and thus the creation of web pages must abide certain rules, let's see:

- a) The page can not contain inclusion of other pages (Example pages of style or scripts).
- b) The page can not contain inclusion of images (jpg or gif file example), any images can be embedded in the page itself.

We see a simple page that displays a presentation message.

Html source page

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>SlimLine - Simple page</title>
</head>
<body>
This page is served by the <b>SlimLine</b>
</body>
</html>
```

Saving the above text in a file, such as **SPage.htm**, and transferring it to the **Storage** directory of the SlimLine, you can see the resulting web page by simply typing in the browser the page address.



Of course, the page can contain links to other pages, it will be possible to achieve a personal navigation between different pages. Here is the same example as before with included the definition of a style.

Html source page

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>SlimLine - Simple page</title>
<style type="text/css">
.Bolded {font-family: Arial, Helvetica, sans-serif; font-size: 20px; font-style: normal; font-weight: bold;}
</style>
</head>
<body>
This page is served by the <span class="Bolded">SlimLine</span>
</body>
</html>
```

9.2 Dinamic web pages

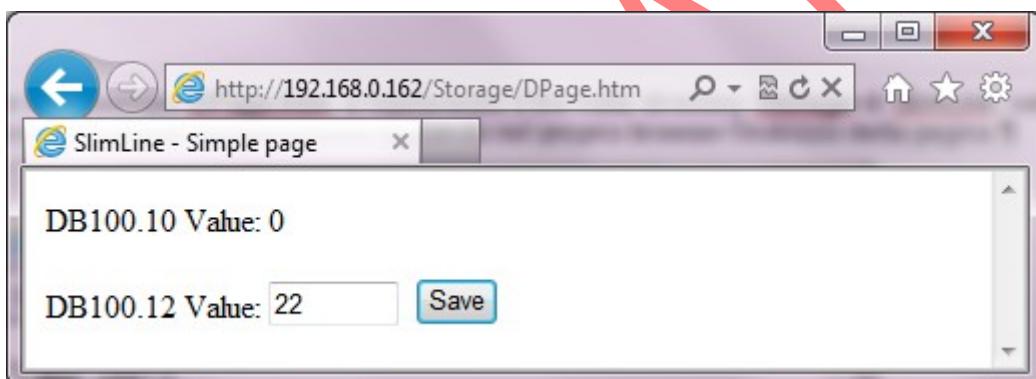
The most important feature of the SlimLine integrated web server is the ability to handle dynamic pages. A dynamic page is a page whose content, in whole or in part, is generated by the server on demand, and therefore can be different each time it is invoked, thus allowing interactivity with the user.

Hence it will be possible to create a page that lists the values of PLC variables and allow you to change the value of them. The following example report an html source of a simple page that displays the value of a PLC variable of type UINT allocated at DB100.10 and allows you to set the value of a PLC variable of type UINT allocated at DB100.12.

Html source page

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>SlimLine - Simple page</title>
</head>
<body>
DB100.10:&nbsp;<!--["%d", UINT, 10]--><br>
<form id="MyForm" name="MyForm" method="post" action="DPage.htm">
DB100.12:&nbsp;<input name="UINT 12" type="text" size="5" maxlength="10" value"<!--["%d", UINT, 12]-->">
&nbsp;<input name="MyButton" type="submit" id="MyButton" value="Save"/>
</form>
</body>
</html>
```

Saving the above text in a file, such as **DPage.htm**, and transferring it to the **Storage** directory of the SlimLine, you can see the resulting web page by simply typing in the browser the page address.



As can be seen, the top row shows the value of the PLC DB100.10 while setting a value in the text box in the bottom row and then pressing the **Save** button, you can set the value of the DB100.12.

Of course in a web page can be displayed and can be set all the desired variables, it is advisable not to go overboard with the number of variables, it is preferable to split them into multiple pages.

9.3 TAGs format

As we have seen in a dynamic page the content is generated on the fly by Http server (SlimLine CPU module), we see what are the mechanisms for defining **TAGs** to be displayed. Within the HTML source of the page you can define a comment fields like <!--["%d", UINT, 10]-->.

The fields are interpreted as comments and therefore can be managed from any HTML editor (Example Macromedia), but the Http server when sending the page to the client (The browser that displays it) replace the field with the value of the variable. In TAG are contained all the necessary information according to the syntax

<!--[Format, Type, Address]-->

9.3.1 Format field

The **Format** string can contain elements of two types, the first consists of characters that are returned to the page unaffected. The second consists in the conversion directives that describe the way in which the topics are to be displayed. The directives conversion begin with a % sign followed by the directives in the format:

% [Flags] [Width] [.Precision] [Length] Conversion

Flags

- + The visualization of signed variables, will always start with the - or + signs.
- space The visualization of signed variables, will always start with the space or – sign.
- x Values other than 0 are prefixed with 0x.
- 0 At the displayed value are added 0 until the desired number of digits (Only for variables of type d, i, o, u, x, X, e, E, f, g, G).

Width: Defines the number of digits to be displayed.

Precision: Defines the number of decimal places to be displayed (Only for variables of type e, E, f).

Length

- h Prima di (d, i, u, x, X, o) denota una variabile short int o unsigned short int.
- I (elle) Prima di (d, i, u, x, X, o) denota una variabile long int o unsigned long int.
- L Prima di (e, E, f, g, G) denota una variabile long double.

Conversion

- d Decimal value with sign.
- i Decimal value with sign.
- o Octal value without sign.
- u Decimal value without sign.
- x Hexadecimal value is displayed using lowercase letters (From 0 to 9, from a to f).
- X Hexadecimal value is displayed using uppercase letters (From 0 to 9, from A to F).
- e Decimal floating-point value, displayed on exponential notation (Example: [-]d.ddde+dd).
- E Decimal floating-point value, displayed on exponential notation (Example: [-]d.dddE+dd).
- f Decimal floating-point value (Example: [-]d.ddd).
- c Single character.
- s String.

9.3.2 Type field

The **Type** field indicates the type of variable you want to display, are managed all types defined in IEC61131.

9.3.3 Address field

The **Address** field indicates the address of the variable, remember that you can specify only variables allocated in the DB 100.

9.3.4 Some TAGs example

To better understand the display format of the TAGs here some examples.

<!--["%d", UINT, 10]-->

Displays the value of a UINT variable allocated at DB 100.10 with a number of integer digits based on the variable value.

<!--["%04d", UINT, 10]-->

Displays the value of a UINT variable allocated at DB 100.10 always expressed with 4 digits.

<!--["%3.0f", REAL, 32]-->

Displays the value of a REAL variable allocated at DB 100.32 with 3 integers digits and no decimal.

<!--["%4.2f", REAL, 50]-->

Displays the value of a REAL variable allocated at DB 100.50 with 2 integers digits and 2 decimals.

DEPRECATE

9.4 ARGs format

The user, other than request to the web server a page can also specify certain parameters to be sent to the web server. To set a PLC variable value from a web page a POST request will be managed, a module is inserted in the web page. When an HTML page contains a **<form>**, the data set on the various objects in the **<form>** are sent so as not to be directly visible to the user, through the HTTP request that the browser sends to the server.

If we refer to the previous example we can see that the part of the HTML page which allows the setting of the PLC variable UINT allocated at DB 100.12 is as follows..

Html source page

```
<form id="MyForm" name="MyForm" method="post" action="DPage.htm">
DB100.12:&nbsp;<input name="UINT 12" type="text" size="5" maxlength="10" value"<!--[%d, UINT, 12]-->">
&nbsp;<input name="MyButton" type="submit" id="MyButton" value="Save"/>
</form>
```

Pratically a **<form>** field with id **MyForm** contains a text box of 5 characters with a maximum of 10 characters with id **UINT12**. In the form is also placed a button of submit type which pressure sends the whole form.

By defining in the browser the value of the text box and pressing the **Save** button, the defined data will be sent to the server that will display the page **DPage.htm** and at the same time it will write the defined value in the **UINT** **DB100.12** variable.

TARG name

The **name** field of the argument is very important, it defines the type of PLC variable to be set (All types defined in IEC61131 are managed) and its address, the two fields must be separated by a space.

A name like **UINT 12** will indicate a **UINT** variable allocated to address **DB 100.12**.

A name like **REAL 128** will indicate a **REAL** variable allocated to address **DB 100.128**.

A name like **STRING 1000 16** will indicate a **STRING** 16 chars length variable, allocated to address **DB 100.1000**.

9.4.1 ARG id

The **id** field of the argument is used to reference the object within the form by the **SetValues()** function. The choice to define it **UINT12** used in the example is just an example, it would be better to use a definition that remind to the meaning (Example "SetPoint", "Preset", etc.).

9.5 Some examples

Of course web pages to be created to reach your needs by entering the desired objects. To facilitate the development of own pages we give the PTP128*000 demonstration program which contains a number of SlimLine programs and related web pages.

To test the various examples, transfer the program on the CPU module with LogicLab and with an FTP client transfer the htm page in the **Storage** directory. Now from a browser you type the IP address of the CPU module followed by the directory and the name of the page Example <http://192.168.0.122/Storage/Page.htm>.

Below are details of some of the examples in the demonstration program..

9.6 LogicIO, logic I/O management

Here's an example of logic I/O management from web page, to view the status of the inputs and set the outputs have been used checkbox objects. The state of active is indicated by the presence of the tick, to activate the outputs set the tick on the desired output and acts on the **Set outputs** button.

To view the actual status of the inputs the page is automatically reload every 10 seconds. To reload the page after the **<head>** directive is placed the statement:

<meta http-equiv="refresh" content="10">

To manage the page some javascript functions are used.

Check(Field, Value), Sets or removes the tick symbol on the checkbox object indicated in **Field** based on **Value**.

SetValues(), Executed at page load it updates all checkbox objects.

SubmitForm(Form), Executed on Set outputs button pressure, it checks if logic output checkbox are set and updates the value of hidden fields that write the PLC variables.

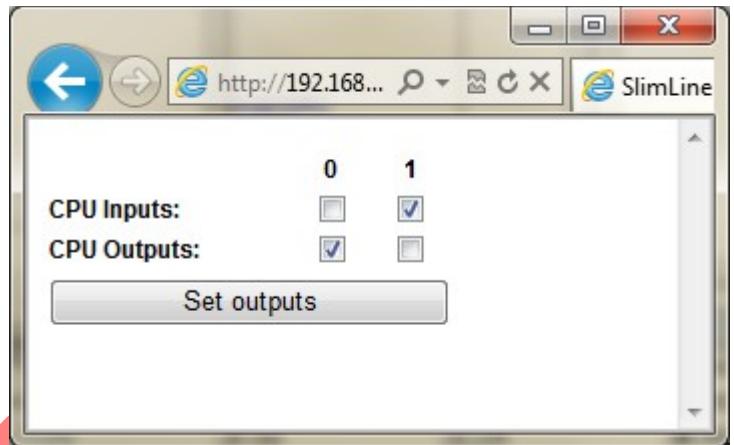
Javascript functions source

```

<script language="javascript">
function Check(Field, Value) {document.MyForm[Field].checked=(Value != 0);}
function SetValues()
{
    Check("Inp00", '<!--[%d, BOOL, 0]-->');
    Check("Inp01", '<!--[%d, BOOL, 1]-->');
    Check("Out00", '<!--[%d, BOOL, 3]-->');
    Check("Out01", '<!--[%d, BOOL, 4]-->');
}

function SubmitForm(Form)
{
    if (document.getElementById('Out00').checked) document.getElementById('BOOL3').value="1";
    if (document.getElementById('Out01').checked) document.getElementById('BOOL4').value="1";
    document.forms[Form].submit();
}
</script>

```



9.7 Updating pages with AJAX

AJAX , acronym for Asynchronous JavaScript and XML, is a software development technique for creating interactive web applications. The development of HTML applications with AJAX is based on an exchange of background data between server and web browser , which enables the dynamic update of a web page without explicitly reloading by the user.

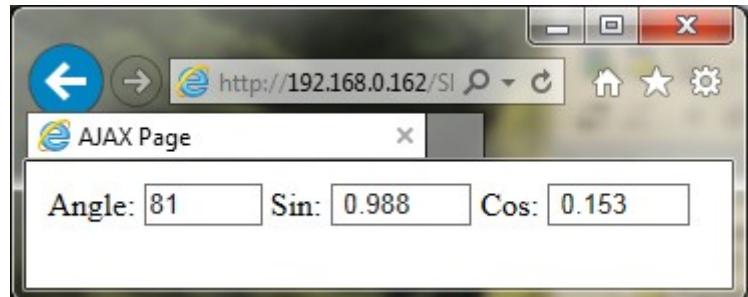
This technique allows for an automatic update of data in a web page without reloading the page, allowing to view the PLC tags automatically. Let's see how this technique works, a java script must be inserted into the web page, it handles AJAX requests . A ready to use script is supplied (Our code SFW191*000).

Here a web page that displays the value of an angle and the respective values of sine and cosine . The values are managed by the SlimLine that executes the change automatically. The values are stored in 3 variables on the DB 100

Angle UINT DB 100.0 It contains the angle

Sin REAL DB 100.4 It contains the sine

Cos REAL DB 100.8 It contains the cosine



html page source

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>AJAX Page</title>
<script type="text/javascript">

// *****
// "SFW191A000" FUNZIONI PER GESTIONE AJAX
// *****

// Le seguenti funzioni gestiscono lo standard AJAX "Asynchronous Java and XML",
// con esse viene gestito lo scambio dinamico di dati con le pagine web.

var XMLHttpRequest=AjaxCreateReqObject();function AjaxCreateReqObject(){var b=null;var
a=navigator.userAgent.toUpperCase();if(window.XMLHttpRequest){b=new
XMLHttpRequest()}else{if(window.ActiveXObject&&(a.indexOf("MSIE 4")<0)){if(a.indexOf("MSIE 5")<0){b=new
ActiveXObject("Msxml2.XMLHTTP")}}else{b=new ActiveXObject("Microsoft.XMLHTTP")}}}return(b)}function
AJAXSendRequest(b){var a=Math.random();if(XMLHttp!=null){XMLHttp.open("GET",b+"?
Rnd="+escape(a),true);XMLHttp.setRequestHeader("connection","close");XMLHttp.onreadystatechange=AJAXHandleR
sp;XMLHttp.send(null)}}function AJAXHandleRsp(){switch(XMLHttp.readyState){case 0:break;case 1:break;case
2:break;case 3:break;case 4:if(XMLHttp.status==200){SetupValues(XMLHttp.responseText)}break}};

// *****
// FUNZIONE "SetupValues(PContent)"
// *****

// Questa funzione viene eseguita su risposta Ajax, nella variabile "PContent"
// è presente tutto il contenuto della pagina richiesta.
// -----
```

```
function SetupValues(PContent)
{
    var Value=new Array(); //Array valori ricevuti da server

    // Eseguo separazione valori, sono separati dal simbolo "|".
    if (PContent.indexOf('|') != -1)
    {
        Value=PContent.split('|');
```

```
document.getElementById("Angle").value=Value[0];
document.getElementById("Sin").value=Value[1];
document.getElementById("Cos").value=Value[2];
}
}

</script>
</head>
<body onLoad="setInterval('AJAXSendRequest('Values.htm')', 3000)">
<table border="0">
<tr>
<td>Angle:</td>
<td><input type="text" id="Angle" size="4" maxlength="4"/></td>
<td>Sin:</td>
<td><input type="text" id="Sin" size="6" maxlength="6"/></td>
<td>Cos:</td>
<td><input type="text" id="Cos" size="6" maxlength="6"/></td>
</tr>
</table>
</body>
</html>
```

On page load `<body onLoad="setInterval('AJAXSendRequest('Values.htm')', 3000)">` the AJAX request of the page **Values.htm** is performed every 3 seconds. The return value of this page is automatically passed to the function **SetupValues** that it parse it and copy the values on the display objects. The page **Values.htm** returns the values of the 3 variables separated by the | symbol. Here's the listing for this page.

Values.htm source page

```
<!--['%d', UINT, 0]-->|<!--['%6.3f', REAL, 4]-->|<!--['%6.3f', REAL, 8]-->
```

10 Tips and tricks

10.1 DWORD variable swap

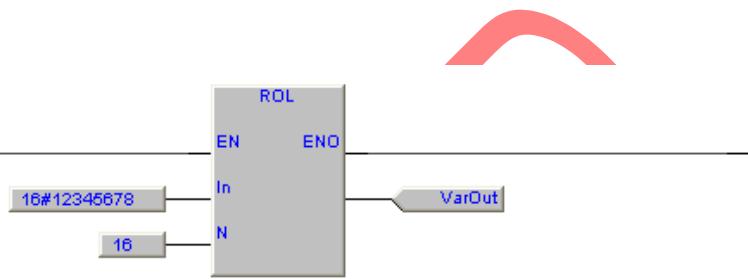
Here's how to use the ROL function to execute the swap of **DWORD** variable.

In the example the value **16#12345678** is converted in the value **16#56781234** and transferred into the variable **VarOut**.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	VarOut	DWORD	Auto	No	0	..	Output variable

LD example



IL example

```

LD 16#12345678
ROL 16
ST VarOut (* Output variable *)
  
```

ST example

```
VarOut:=ROL(16#12345678, 16); (* Output variable *)
```

10.2 WORD variable swap

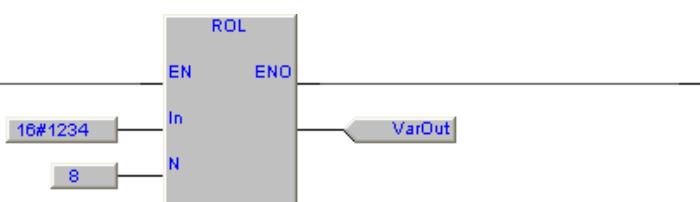
Here's how to use the ROL function to execute the swap of **WORD** variable.

In the example the value **16#1234** is converted in the value **16#3412** and transferred into the variable **VarOut**.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	VarOut	WORD	Auto	No	0	..	Output variable

LD example



IL example

```

LD 16#1234
ROL 8
ST VarOut (* Output variable *)
  
```

ST example

```
VarOut:=ROL(16#1234, 8); (* Output variable *)
```

10.3 BYTE variable swap

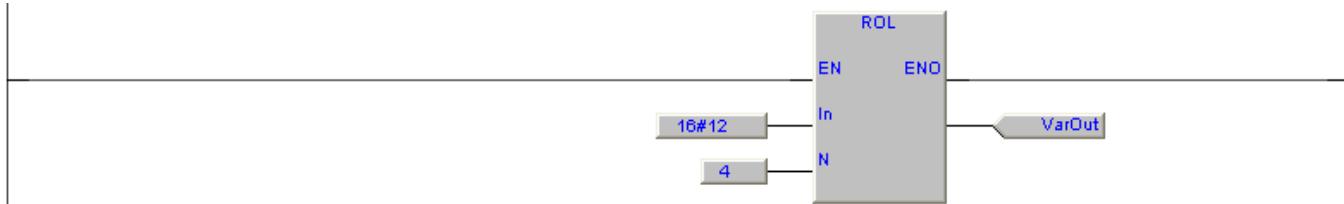
Here's how to use the ROL function to execute the swap of **BYTE** variable.

In the example the value **16#12** is converted in the value **16#21** and transferred into the variable **VarOut**.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	VarOut	BYTE	Auto	No	0	..	Outputvariable

LD example



IL example

```

LD 16#12
ROL 4
ST VarOut (* Output variable *)
  
```

ST example

```
VarOut:=ROL(16#12, 4); (* Output variable *)
```

DEPRECATE

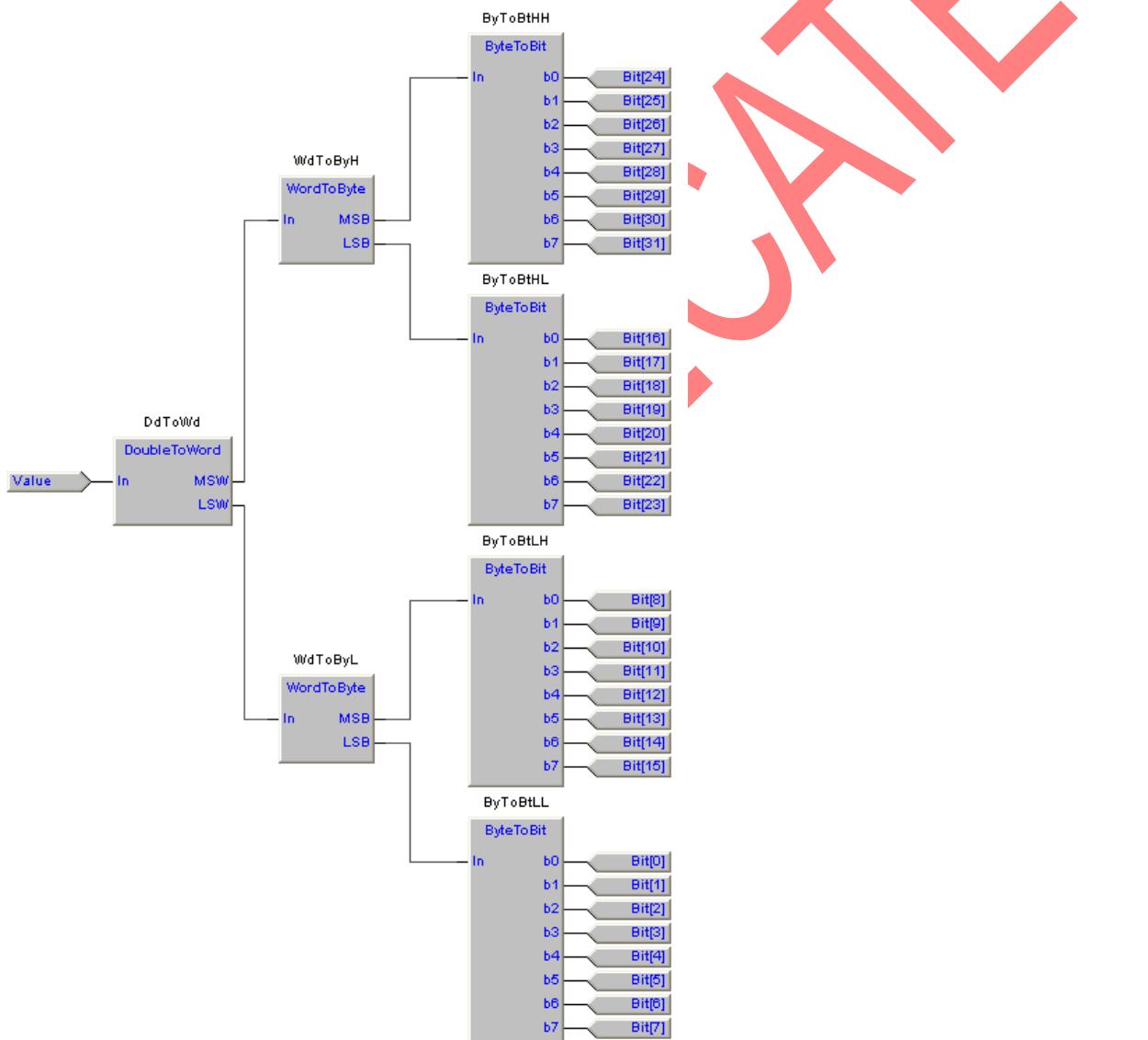
10.4 Expand DWORD to 32 BOOL

Here's how to using the ***DoubleToWord***, ***WordToByte***, ***ByteToBit*** function blocks, it's possible to expand a ***DWORD*** variable to 32 ***BOOL*** variables.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	ByToBtHH	ByteToBit	Auto	No	0	..	Byte to bits
2	ByToBtHL	ByteToBit	Auto	No	0	..	Byte to bits
3	ByToBtLH	ByteToBit	Auto	No	0	..	Byte to bits
4	ByToBtLL	ByteToBit	Auto	No	0	..	Byte to bits
5	WdToByH	WordToByte	Auto	No	0	..	Word to bytes
6	WdToByL	WordToByte	Auto	No	0	..	Word to bytes
7	DdToWd	DoubleToWord	Auto	No	0	..	Double to word
8	Bit	BOOL	Auto	[0..31]	FALSE,31(0)	..	Bit status
9	Value	DWORD	Auto	No	0	..	Value to convert

FBD example (Ptp114a200, FBD_DWExpand)



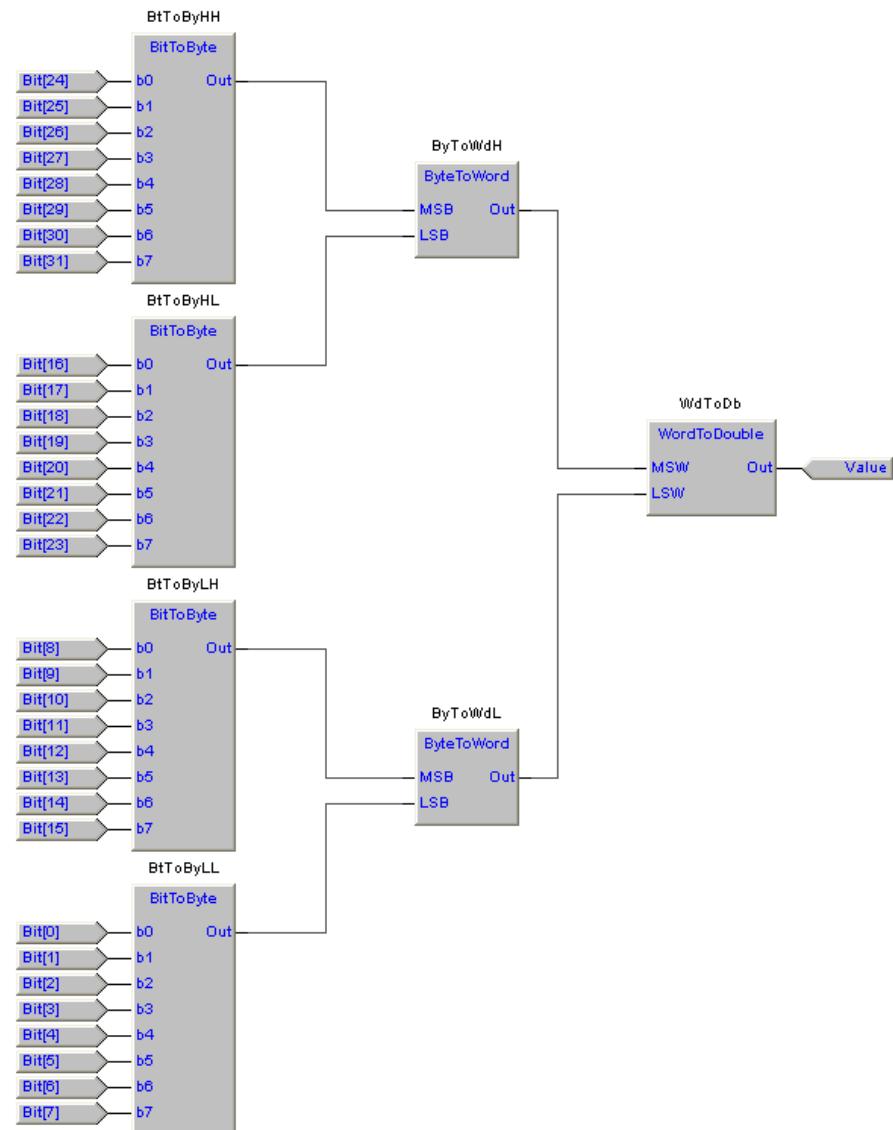
10.5 Compress 32 BOOL into a DWORD

Here's how to using the [BitToByte](#), [ByteToWord](#), [WordToDouble](#) function blocks it's possible to compress 32 **BOOL** variables into a single **DWORD** variable.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
2	BtToByHL	BitToByte	Auto	No	0	..	Bits to byte
3	BtToByLH	BitToByte	Auto	No	0	..	Bits to byte
4	BtToByLL	BitToByte	Auto	No	0	..	Bits to byte
5	ByToWdH	ByteToWord	Auto	No	0	..	Byte to word
6	ByToWdL	ByteToWord	Auto	No	0	..	Byte to word
7	WdToDb	WordToDouble	Auto	No	0	..	Word to double
8	Value	DWORD	Auto	No	0	..	Value to convert
9	Bit	BOOL	Auto	[0..31]	FALSE,31(0)	..	Bit status

FBD example (Ptp114a200, FBD_DWCompress)



REMOVED

10.6 Define non-printable ascii characters

In the management of communication protocols and/or to set print mode on printers, usually need to output non-printable ascii characters, ie characters with codes less than 16#20 or greater than 16#7F.

For a definition of printable ascii characters just include characters to single quotes (Example '**abcd**').

For non printable characters, you must precede the hexadecimal value of the character with the \$ sign, so to define the <STX> 16#02 character, we use '\$02', for <ETX> '\$03' and so on.

Please remember that for some control characters such as line feed, code 16#0A, it's possible to define it as '\$0A' or as '\$l'. The carriage return, code 16#0D, can be defined as either '\$0D' or '\$r'. Please refer to the table.

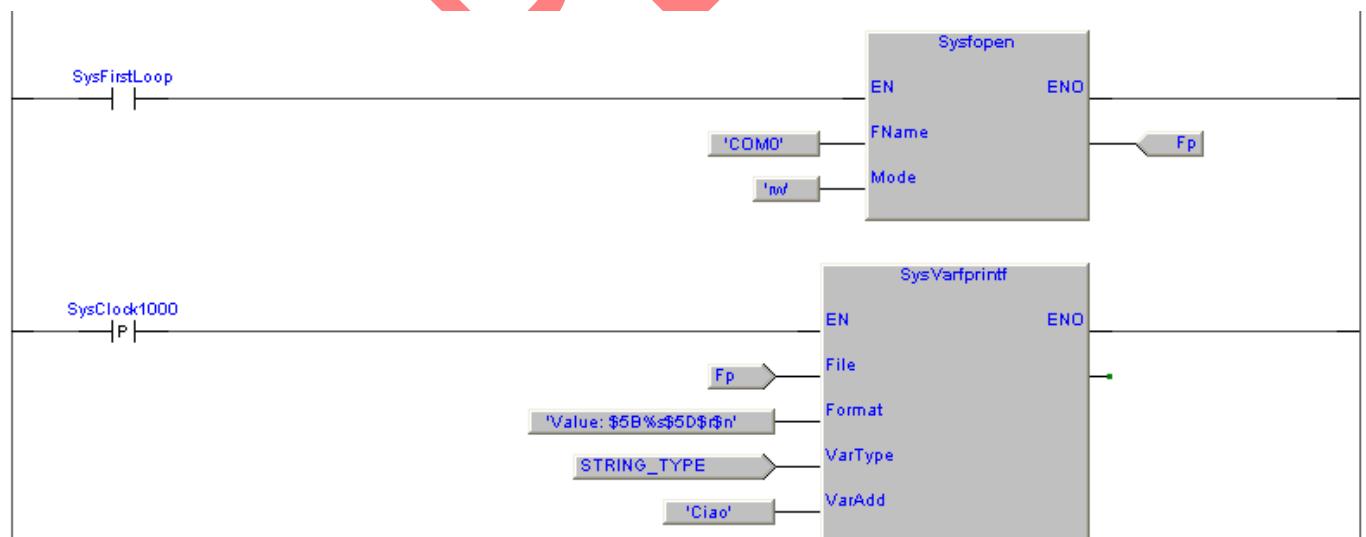
Sequence	Meaning	Hex value	Example
\$\$	Character \$	16#24	'I paid \$\$5 for this'
\$'	Apostrophe	16#27	'Enter \$'Y\$' for YES'
\$l	Line feed	16#0A	'next \$l line'
\$r	Carriage return	16#0D	'Hello\$r'
\$n	New line	16#0D0A	'This is a line\$n'
\$p	New page	16#0C	'last line \$p first line'
\$t	Tabulation	16#09	'name\$tsize\$tdate'
\$hh		16#hh	'ABCD = \$41\$42\$43\$44'

Here's an example of using the **SysVarprintf** function to define printable characters and non printable characters, and send them to the output stream. In this example, is sent to the serial port COM0 the **[Ciao]** string followed by carriage return and line feed.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	Fp	FILEP	Auto	No	0	..	File pointer

LD example



10.7 Rx/Tx data to stream

As seen, with the **Sysopen** function it's possible to define a link between an I/O resource and a flow of data **stream**, from which you can manage the reception and/or transmission of data.

To receive the input data from the stream the **SysGetIChars** function is used to check if any data is available and the **Sysfgetc** function to read them.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	RxString	STRING	Auto	[32]		..	Rx string
2	File	FILEP	Auto	No	0	..	File pointer
3	Ptr	@USINT	Auto	No	0	..	String pointer

ST example

```
(* Rx data from stream. *)

Ptr:=ADR(RxString); (* String pointer *)

WHILE (TO_BOOL(SysGetIChars(File))) DO
    @Ptr:=TO_USINT(Sysfgetc(File)); (* Rx string *)
    Ptr:=Ptr+1; (* String pointer *)

    (* Check if string pointer overflow. *)

    IF (Ptr > ADR(RxString)+31) THEN EXIT; END_IF;
END WHILE;
```

To the data transmission to the stream, the **SysGetOSpace** function is used to check if there is enough space to hold the string, or as in the example, if the output buffer is empty, then it's possible to send the string with the **SysVarprintf** function.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	TxString	STRING	Auto	[32]		..	Rx string
2	File	FILEP	Auto	No	0	..	File pointer
3	i	UINT	Auto	No	0	..	Auxiliary counter

ST example

```
(* Tx data to stream. *)

IF (TO_UDINT(SysGetOSpace(File)) = SysGetTxBSIZE(File)) THEN
    i:=TO_UINT(SysVarprintf(File, '%s', STRING_TYPE, ADR(TxString)));
END_IF;
```

10.8 Data types conversion

The data conversion (Also known as **casting**) is a necessary practice in programming, of course, if the target data type has a lower range of the source data type the value is truncated. Let's see the various conversions:

BOOL Type: Conversions from any type to a **BOOL**, return FALSE if the data value to be converted is 0. Return TRUE if the data value to be converted is different from 0 (also <0).

SINT/USINT Type: Conversions from any type to a **USINT**, return the value of the least significant byte of the value to convert expressed in hexadecimal. Example, the value 4660 (16 # 1234) will return 52 (16 # 34), the same goes for REAL example 300.0 (16 # 012C) will return 44 (16 # 2C). For the type **SINT** if the hexadecimal value exceeds 127, the number is negative.

INT/UINT Type: Conversions from any type to a **UINT** return the value of the two least significant bytes of the value to convert expressed in hexadecimal. Example, the value 305419896 (16 # 12345678) will return 22 136 (16 # 5678), the same applies to the REAL example 90000.0 (16 # 15F90) will return to 24 464 (16 # 5F90). For the **INT** if the hexadecimal value exceeds 32767 the number is negative.

In the LogicLab IEC programming, special functions are provided to manage data type conversion, let's see.

Name	Input type	Output type	Function
DINT_TO_INT	DINT	INT	Converts a long integer (32 bits, signed) into an integer (16 bits, signed)
INT_TO_DINT	INT	DINT	Converts an integer (16 bits, signed) into a long integer (32 bits, signed)
TO_BOOL	All	BOOL	Converts All data type into a boolean
TO_SINT	All	SINT	Converts All data type into a short integer (8 bits, signed)
TO_USINT	All	USINT	Converts All data type into an unsigned short integer (8 bits, unsigned)
TO_INT	All	INT	Converts All data type into an integer (16 bits, signed)
TO_UINT	All	UINT	Converts All data type into an unsigned integer (16 bits, unsigned)
TO_DINT	All	DINT	Converts All data type into a long integer (32 bits, signed)
TO_UDINT	All	UDINT	Converts All data type into an unsigned long integer (32 bits, unsigned)
TO_REAL	All	REAL	Converts All data type into a floating point (32 bits, signed)

Example

Convert a variable DINT variable to a USINT variable in different programming languages. Of course, if the **VarDINT** variable value exceeds the value 255 (Limit of the **VarUSINT** variable), the rest of the division for the value and the limit will be returned.

Defining variables

	Name	Type	Address	Array	Init value	Attribute	Description
1	VarUSINT	USINT	Auto	No	0	..	USINT variable
2	VarDINT	DINT	Auto	No	0	..	DINT variable

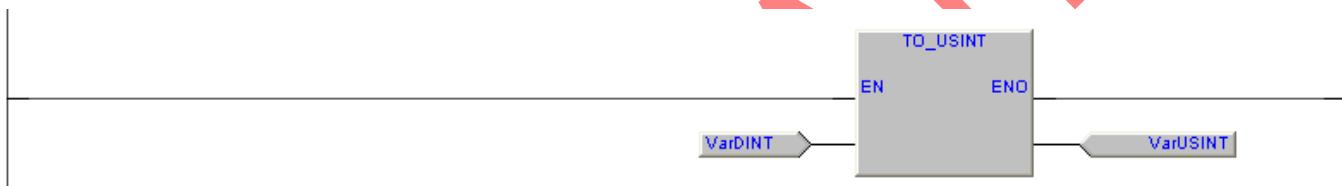
IL example

```
LD  VarDINT (* DINT variable *)
TO_USINT
ST  VarUSINT (* USINT variable *)
```

FBD example



LD example



ST example

```
VarUSINT:=TO_USINT(VarDINT); (* USINT variable *)
```

10.9 User Informations and Settings

E' previsto un interfacciamento tra il programma utente PLC ed il sistema operativo, questo permette di visualizzare ed impostare da sistema operativo variabili il cui valore è disponibile all'interno del programma utente.

There's a user interface between the PLC user program and the operating system, this allows to view and set by the operating system, variables, whose value is available in the user program.

SysUInfoA, SysUInfoB, SysUInfoC, SysUInfoD, 4 string variables of 16 characters each, which can be set by the PLC user program and viewed by the operating system.

SysUSetA, SysUSetB, SysUSetC, SysUSetD, 4 string variables of 16 characters each, which can be set by the operating system and read by the PLC user program.

There's a web page (**User** menu) which displays the **SysUInfo(x)** variables and allows to set the **SysUSet(x)** variables.

ELSiST - SlimLine

[Home](#) | [General Setup](#) | [Time Setup](#) | [User](#)

User informations and settings

(x)	SysUInfo(x)	SysUSet(x)
A	12	12
B	34	34
C	56	56
D	78	78

Read
Write

ELSIST - SlimLine
 For more information visit our website: www.elsist.it

In the example a simple program that reads the values of variables **SysUSet(x)** and transfers them to the respective **SysUInfo(x)** variable. In the web page, the value set in the **SysUSet(x)** variable, on acceptance with the Write button, is returned in the **SysUInfo(x)** variable, as shown in the screenshot above.

Definizione variabili

	Name	Type	Address	Array	Init value	Attribute	Description
1	Values	UDINT	Auto	[0..3]	4(0)	..	User variables value
2	i	INT	Auto	No	0	..	Auxiliary variable

Esempio ST

```
(* Read user settings and write user infos. *)

IF (SysUVSet) THEN
    i:=SysVarsscanf(ADR(SysUSetA), '%d', UDINT_TYPE, ADR(Values[0]));
    i:=SysVarsnprintf(ADR(SysUInfoA), 16, '%d', UDINT_TYPE, ADR(Values[0]));

    i:=SysVarsscanf(ADR(SysUSetB), '%d', UDINT_TYPE, ADR(Values[1]));
    i:=SysVarsnprintf(ADR(SysUInfoB), 16, '%d', UDINT_TYPE, ADR(Values[1]));

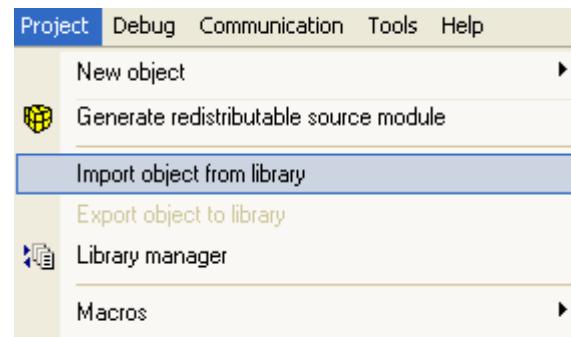
    i:=SysVarsscanf(ADR(SysUSetC), '%d', UDINT_TYPE, ADR(Values[2]));
    i:=SysVarsnprintf(ADR(SysUInfoC), 16, '%d', UDINT_TYPE, ADR(Values[2]));

    i:=SysVarsscanf(ADR(SysUSetD), '%d', UDINT_TYPE, ADR(Values[3]));
    i:=SysVarsnprintf(ADR(SysUInfoD), 16, '%d', UDINT_TYPE, ADR(Values[3]));
END_IF;
```

11 Programming examples

11.1 Example library

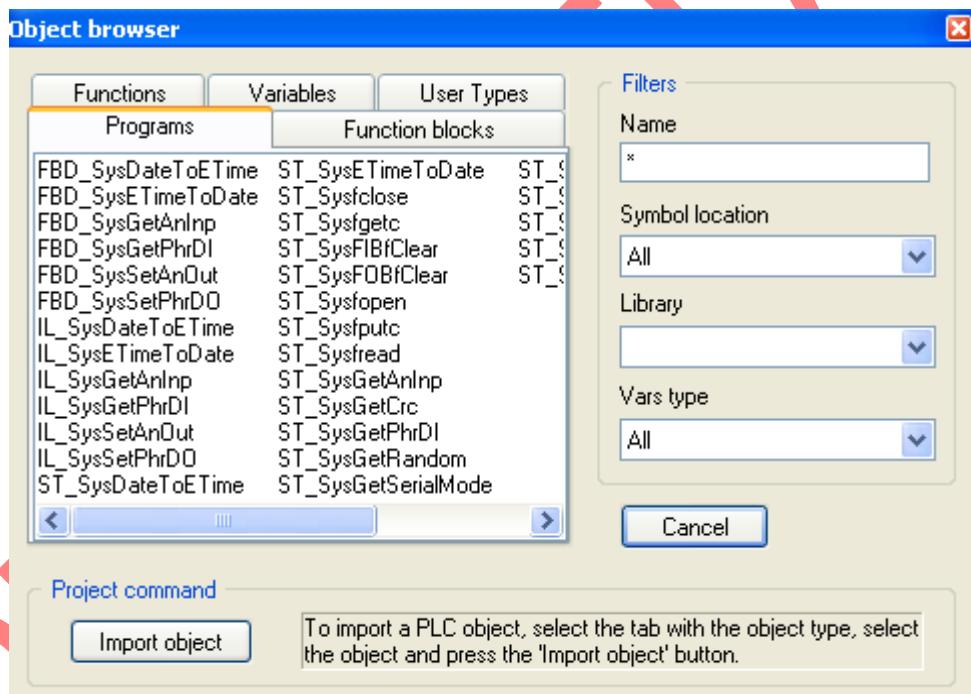
To allow the user to have examples how develop their own programs, almost all the examples in the manual are provided with demonstration program. The demonstration programs are coded with the suffix PTP and are reported next to each example. If you want to include in your project a sample file of the manual, from the **Project** menu select the item **Import object from library**. A dialog box will be open, it allows to select the library from which to extract the program to import.



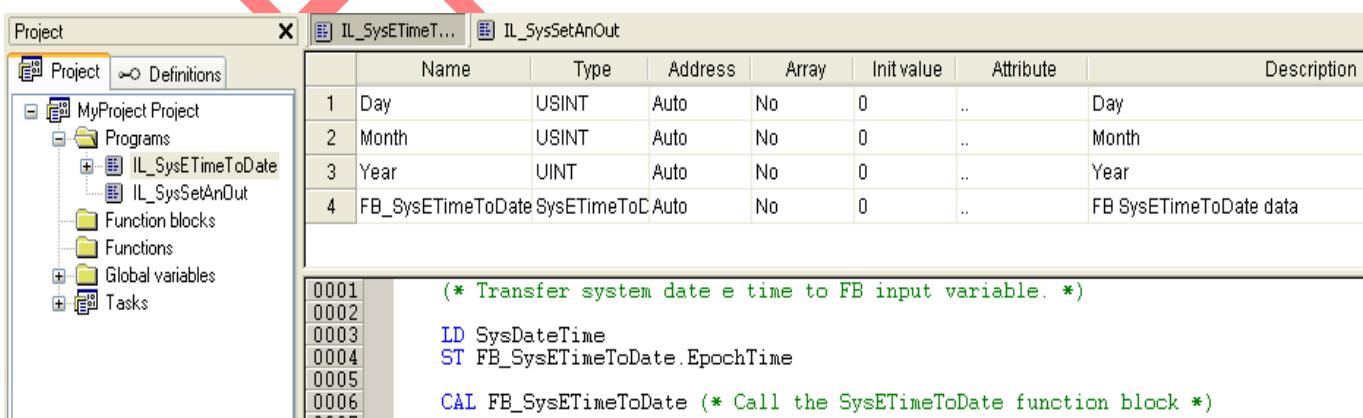
Choose the desired library file (Example Ptp116*000.dll *), a dialog box with a list of objects will open, the desired object can be selected from it.

By highlighting the objects and then pressing the **Import Object** button, the selected objects will be included in your project.

In addition to the programs, also variables, can be imported from the library. In this way you can import all the logic I/O definitions as shown in the [definition table](#).



Once included in the project the samples, it will be possible to use them directly, or, with simple cut and paste operations, pasting part of the source code from the sample.



11.2 Logic I/O definitions in the examples

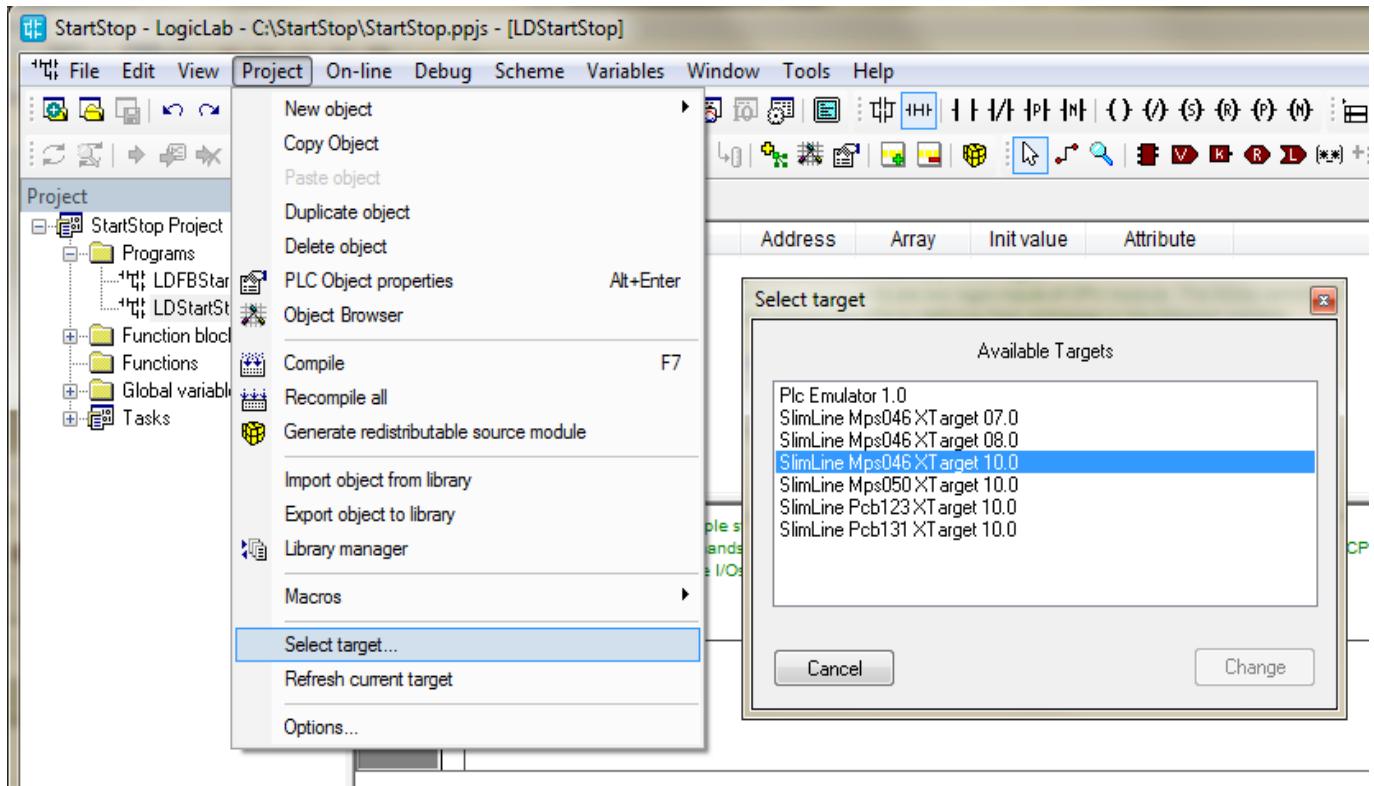
All examples in this manual have been made by using a system configured with a MPS046*100 SlimLine CPU module combined with a PCB122*100 Mixed I/O module (Set to address 0). All I/Os module were combined with mnemonic variables. The inputs are called **Di0xM00** the outputs **Do0xM00** as reported in the definition table.

	Name	Type	Address	Group	Array	Init value	Attribute	Description
1	Di00M00	BOOL	%IX0.0		No	FALSE	..	Input 00, Module address 0
2	Di01M00	BOOL	%IX0.1		No	FALSE	..	Input 01, Module address 0
3	Di02M00	BOOL	%IX0.2		No	FALSE	..	Input 02, Module address 0
4	Di03M00	BOOL	%IX0.3		No	FALSE	..	Input 03, Module address 0
5	Di04M00	BOOL	%IX0.4		No	FALSE	..	Input 04, Module address 0
6	Di05M00	BOOL	%IX0.5		No	FALSE	..	Input 05, Module address 0
7	Di06M00	BOOL	%IX0.6		No	FALSE	..	Input 06, Module address 0
8	Di07M00	BOOL	%IX0.7		No	FALSE	..	Input 07, Module address 0
9	Di08M00	BOOL	%IX0.8		No	FALSE	..	Input 08, Module address 0
10	Di09M00	BOOL	%IX0.9		No	FALSE	..	Input 09, Module address 0
11	Di10M00	BOOL	%IX0.10		No	FALSE	..	Input 10, Module address 0
12	Di11M00	BOOL	%IX0.11		No	FALSE	..	Input 11, Module address 0
13	Do00M00	BOOL	%QX0.0		No	FALSE	..	Output 00, Module address 0
14	Do01M00	BOOL	%QX0.1		No	FALSE	..	Output 01, Module address 0
15	Do02M00	BOOL	%QX0.2		No	FALSE	..	Output 02, Module address 0
16	Do03M00	BOOL	%QX0.3		No	FALSE	..	Output 03, Module address 0
17	Do04M00	BOOL	%QX0.4		No	FALSE	..	Output 04, Module address 0
18	Do05M00	BOOL	%QX0.5		No	FALSE	..	Output 05, Module address 0
19	Do06M00	BOOL	%QX0.6		No	FALSE	..	Output 06, Module address 0
20	Do07M00	BOOL	%QX0.7		No	FALSE	..	Output 07, Module address 0

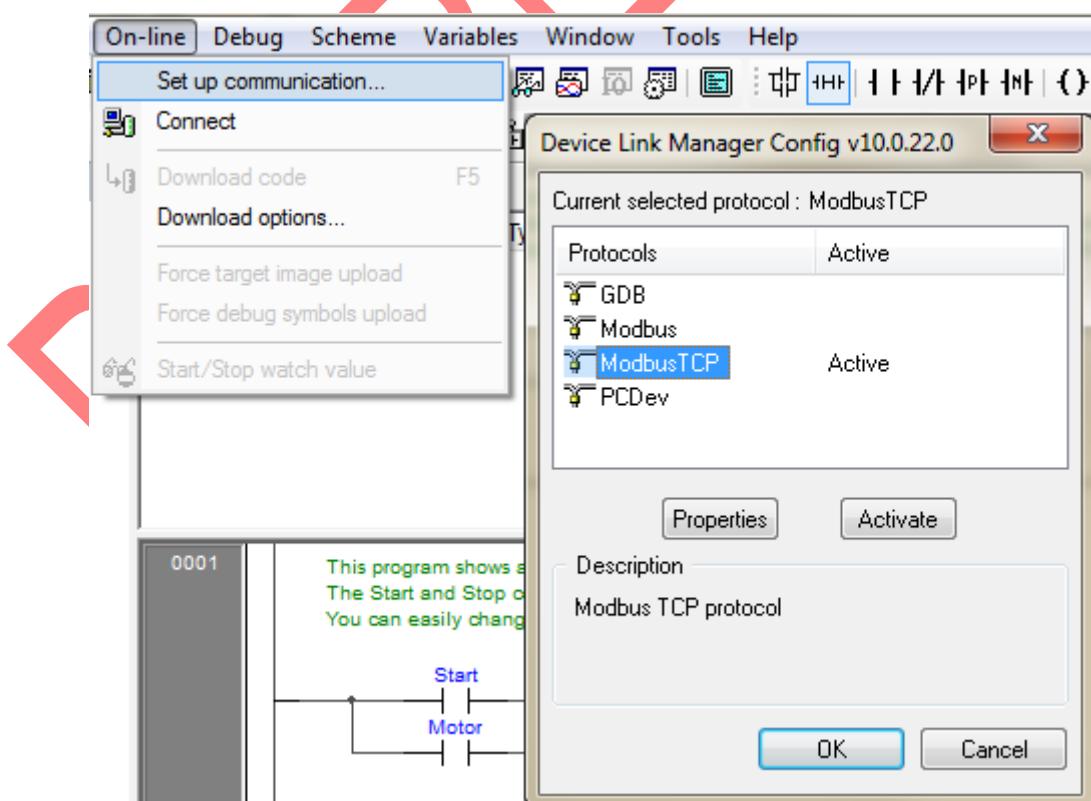
DEPRECATE

11.3 Examples provided with LogicLab

In the LogicLab distribution are inserted some example programs, the programs are provided in source code and can be transferred to the target system and tested. In the LogicLab main window, under the Example projects there are link to the various projects. To use an example on your target system, the system with which you are working must be defined from the menu **Project → Select target**.



Then you must define the communication mode from the menu **On-line → Set up communication**.



11.3.1 Example programs list

The example program released with the LogicLab are.

StartStop	Start/Stop logic
SMSbyWeb	Sending SMS messages from web page
MdbAsciiMaster	FB to manage the Modbus ASCII protocol (Master Mode)
MultipleSTE	SNMP connection with some STE devices
EasyProtocol	How to develop a simple communication protocol
PowerOneCm	Communication with Power One Aurora inverter
TagReader	Access control by using a I button TAG
GSMDoorOpen	Door open with a free pone call
TCPAsciiProtocol	Simple ascii communication on TCP/IP
CSVFileTimeSwitch	Programmable timer with time reading from CSV files
SineWave	Sine wave generator

DEPRECATE

12 Appendix

12.1 IL instructions table

Instruction	Operands	Description
LD	All	Loads the operand value in the accumulator
LDN	All	Loads the (NOT) operand value in the accumulator
ST	All	Stores the accumulator value into operand location
STN	All	Stores the (NOT) accumulator value into operand location
S	BOOL	Sets operand to TRUE if accumulator is TRUE
R	BOOL	Sets operand to FALSE if accumulator is TRUE
AND	All but REAL	Bitwise AND between accumulator and operand, result in accumulator
ANDN	All but REAL	Bitwise AND between accumulator and (NOT) operand, result in accumulator
OR	All but REAL	Bitwise OR between accumulator and operand, result in accumulator
ORN	All but REAL	Bitwise OR between accumulator and (NOT) operand, result in accumulator
XOR	All but REAL	Bitwise XOR between accumulator and operand, result in accumulator
XORN	All but REAL	Bitwise XOR between accumulator and (NOT) operand, result in accumulator
NOT		Accumulator value bitwise inversion
ADD	All but BOOL	Sum between accumulator and operand, result in accumulator
SUB	All but BOOL	Subtraction between accumulator and operand, result in accumulator
MUL	All but BOOL	Multiplication between accumulator and operand, result in accumulator
DIV	All but BOOL	Division between accumulator and operand, result in accumulator
MOD	All but BOOL	Return the module of division in the accumulator
GT	All but BOOL	Checks if accumulator > operand, result (BOOL) in the accumulator
GE	All but BOOL	Checks if accumulator >= operand, result (BOOL) in the accumulator
EQ	All but BOOL	Checks if accumulator = operand, result (BOOL) in the accumulator
NE	All but BOOL	Checks if accumulator <> operand, result (BOOL) in the accumulator
LE	All but BOOL	Checks if accumulator <= operand, result (BOOL) in the accumulator
LT	All but BOOL	Checks if accumulator < operand, result (BOOL) in the accumulator
JMP	Label	Jump to label unconditionally
JMPC	Label	Jump to label if accumulator is different from 0
JMPCN	Label	Jump to label if accumulator is equal to 0
CAL	FB	Execute the function block unconditionally
CALC	FB	Execute the function block if accumulator is different from 0
CALCN	FB	Execute the function block if accumulator is equal to 0
RET		Return unconditionally to the program that executed CALL
RETC		Return to the program that executed CALL if accumulator is different from 0

12.2 ST language operators

The following table shows the operators used in the ST language. The operators are listed in the table due to their priority, from top to bottom, so the brackets have higher priority over all other operators.

Operation	Symbol	Example
Parenthesization	(Expression)	
Function evaluation	Function(Arguments)	LN(A), MAX(X,Y), etc.
Negation	-	
Complement	NOT	
Exponentiation	**	
Multiply	*	
Divide	/	
Modulo	MOD	
Add	+	
Subtract	-	
Comparison	<, >, <=, >=	
Equality	=	
Inequality	<>	
Boolean AND	&	
Boolean AND	AND	
Boolean Exclusive OR	XOR	
Boolean OR	OR	

DEPRECATED

12.3 ST language statements

The following table shows the operators used in the ST language. The operators are listed in the table due to their priority, from top to bottom, so the brackets have higher priority over all other operators.

Statement	Example
Assignment	A:=B; CV:=CV+1; C:=SIN(X);
FB Invocation and output usage	CMD_TMR(IN:=%IX5, PT:=T#300ms); A:=CMD_TMR.Q;
RETURN	RETURN;
IF	D:=B*B-4*A*C; IF D < 0.0 THEN NROOTS:=0; ELSIF D=0.0 THEN NROOTS:=1; X1:=-B/(2.0*A); ELSE NROOTS:=2; X1:=(-B+SQRT(D))/(2.0*A); X2:=(-B-SQRT(D))/(2.0*A); END_IF;
CASE	TW:=BCD_TO_INT(THUMBWHEEL); TW_ERROR:=0; CASE TW OF 1,5: DISPLAY:=OVEN_TEMP; 2: DISPLAY:=MOTOR_SPEED; 3: DISPLAY:=GROSS-TARE; 4,6..10: DISPLAY:=STATUS(TW-4); ELSE DISPLAY:=0; TW_ERROR:=1; END_CASE; QW100:=INT_TO_BCD(DISPLAY);
FOR	J:=101; FOR I:=1 TO 100 BY 2 DO IF WORDS[I]='KEY' THEN J:=I; EXIT; END_IF; END_FOR;
WHILE	J:=1; WHILE J <= 100 & WORDS[J]<>'KEY' DO J:=J+2; END WHILE;
REPEAT	J:=-1; REPEAT J:=J+2; UNTIL J=101 OR WORDS[J]='KEY' END_REPEAT ;
EXIT	EXIT;
Empty Statement	;

12.4 Object IDs

All objects (Functions and / or function blocks) have a unique identifier that distinguishes them. The ID is also returned in the eventual error code (Reference to the [SysGetLastError](#) function and the [SysLastError](#) variable). The error code is composed by the object ID followed by digits indicating the error code.

The following table (Ref. MDP070xx00) reports all the object Ids, the symbol (●) indicates that the object is obsolete. On the obsolete objects are shown (In brackets) the latest version of the library and manual in which the object was present.

ID	Object
9931 (000 ÷ 999)	SysFWVarsscanf , extracts values from string with find
9932 (000 ÷ 999)	SysLWVarsprintf , variable print to string with append
9933 (000 ÷ 999)	SysRMAlloc , allocates the relocatable memory
9934 (000 ÷ 999)	SysRmFree , frees the relocatable memory
9935 (000 ÷ 999)	SysGetEndianness , get the system endianness
9936 (000 ÷ 999)	SysGetUTCDate Time , get the system UTC date and time
9937 (000 ÷ 999)	SysSetUTCDate Time , set the system UTC date and time
9938 (000 ÷ 999)	SysCalcCrc , CRC calculation
9939 (000 ÷ 999)	SysFIisOpen , get the file open status
9940 (000 ÷ 999)	SysTimeZoneAdj , adjust date and time
9941 (000 ÷ 999)	SysTCPClient , opens a TCP/IP connection
9942 (000 ÷ 999)	SysTCPServer , accept TCP/IP connections
9943 (000 ÷ 999)	SysUDPClient , send UDP data stream
9944 (000 ÷ 999)	SysUDPServer , accept UDP data stream
9945 (000 ÷ 999)	SysGetIpInfos , returns IP infos
9946 (000 ÷ 999)	SysSerialPort , manage serial port
9947 (000 ÷ 999)	SysMAlloc , memory allocation
9948 (000 ÷ 999)	SysSetTaskLpTime , set task loop time
9949 (000 ÷ 999)	Funzione embedded
9950 (000 ÷ 999)	SysSpyData , system spy data
9951 (000 ÷ 999)	SysSetPWMOut , set PWM output
9952 (000 ÷ 999)	SysDirListing , directory listing
9953 (000 ÷ 999)	SysI2CWrd , writes/reads on I2C extension bus
9954 (000 ÷ 999)	SysCANTxMsg , transmit a CAN message
9955 (000 ÷ 999)	SysCANRxMsg , receives a CAN message
9956 (000 ÷ 999)	SysIsCANRxTxAv , checks if CAN Rx or Tx is available
9957 (000 ÷ 999)	SysCANSetMode , set the CAN controller mode
9958 (000 ÷ 999)	Sysfseek , file seek
9959 (000 ÷ 999)	Sysfilelength , file length
9960 (000 ÷ 999)	Sysrename , file rename
9961 (000 ÷ 999)	Sysremove , file remove
9962 (000 ÷ 999)	SysFOBfFlush , file output buffer flush
9963 (000 ÷ 999)	SysFOBfClear , file output buffer clear
9964 (000 ÷ 999)	SysFIBfClear , file input buffer clear
9965 (000 ÷ 999)	SysFGetOBfSize , get file Tx output buffer size
9966 (000 ÷ 999)	SysFGetIBfSize , get file Rx input buffer size
9967 (000 ÷ 999)	SysFGetOSpace , get output available space on file
9968 (000 ÷ 999)	SysFGetIChars , get input available characters from file
9969 (000 ÷ 999)	Sysfwrite , write data to file
9970 (000 ÷ 999)	Sysfread , read data from file
9971 (000 ÷ 999)	Sysputc , put character to file

- 9972 (000 ÷ 999) [Sysfgetc](#), get character from file
 9973 (000 ÷ 999) [Sysfclose](#), file close
 9974 (000 ÷ 999) [SysIPReach](#), IP address is reachable
- 9975 (000 ÷ 999) [SysUDPSktRcv](#), UDP socket receive (XTarget_11_0, MNL151C180)
 - 9976 (000 ÷ 999) [SysUDPSktSend](#), UDP socket send (XTarget_11_0, MNL151C180)
 - 9977 (000 ÷ 999) [SysSktListen](#), socket listen (XTarget_11_0, MNL151C180)
 - 9978 (000 ÷ 999) [SysGetCrc](#), get CRC value (XTarget_11_0, MNL151C180)
- 9979 (000 ÷ 999) [SysDMXMng](#), DMX management
 9980 (000 ÷ 999) [SysGetEncoder](#), get encoder input
 9981 (000 ÷ 999) [SysGetCounter](#), get counter
 9982 (000 ÷ 999) [SysSetAnOut](#), set analog output
 9983 (000 ÷ 999) [SysGetAnInp](#), get analog input
 9984 (000 ÷ 999) [SysSetPhrDO](#), set peripheral digital output
 9985 (000 ÷ 999) [SysGetPhrDI](#), get peripheral digital input
 9986 (000 ÷ 999) [SysETimeToDate](#), epoch time to date conversion
 9987 (000 ÷ 999) [SysDateToETime](#), date to epoch time conversion
 9988 (000 ÷ 999) [SysPhrVWr](#), write variable to peripheral module
 9989 (000 ÷ 999) [SysPhrVRd](#), read variable from peripheral module
 9990 (000 ÷ 999) [SysPhrInfos](#), get infos from peripheral modules
 9991 (000 ÷ 999) [SysPCodeAccept](#), accepts the protection code
 9992 (000 ÷ 999) [SysSetSerialDTR](#), set DTR signal status
 9993 (000 ÷ 999) [SysGetSerialCTS](#), get serial CTS signal status
 9994 (000 ÷ 999) [SysSetSerialMode](#), set serial mode (XTarget_11_0, MNL151C180)
 9995 (000 ÷ 999) [SysGetSerialMode](#), get serial mode
 9996 (000 ÷ 999) [Sysfopen](#), file open
 9997 (000 ÷ 999) [SysVarsprintf](#), variable print to string
 9998 (000 ÷ 999) [SysVarprintf](#), variable print to file
 9999 (000 ÷ 999) [SysVarsscanf](#), extracts values from string
- 10000 (000 ÷ 999) MDBRTUMASTER, modbus Rtu master (ePLCUtyLib_A400, MNL151B040)
 - 10001 (000 ÷ 999) [CPUModuleIO](#), CPU module I/O management (ePLCUtyLib_B000, MNL151C180)
- 10002 (000 ÷ 999) [ModemCore_v4](#), modem core management
 10003 (000 ÷ 999) [ModemSMSReceive](#), receive a SMS message
 10004 (000 ÷ 999) [ModemSMSSRxCmd_v1](#), receive a SMS command
 10005 (000 ÷ 999) [ModemSMSSend_v2](#), send a SMS message
- 10006 (000 ÷ 999) [SetSMode](#), Set serial mode (ePLCUtyLib_B000, MNL151C180)
- 10007 (000 ÷ 999) [ModbusMaster](#), modbus RTU master
 10008 (000 ÷ 999) [OWireMng](#), One-Wire management
 10009 (000 ÷ 999) [OWRdIdentifier](#), One-Wire read ROM identifier
 10010 (000 ÷ 999) [OWRdTTemperature](#), One-Wire read temperature
 10011 (000 ÷ 999) [IODataExchange](#), exchange data by using logic I/O
 10012 (000 ÷ 999) [PIDMng](#), PID management
 10013 (000 ÷ 999) [STESnmpAcq](#), STE termometer acquisition over SNMP
 10014 (000 ÷ 999) [UDPDataTxfer](#), UDP data transfer
 10015 (000 ÷ 999) [OWRdHumidity](#), One-Wire read humidity
 10016 (000 ÷ 999) [IEC62056_21Rd](#), IEC62056-21 protocol read
 10017 (000 ÷ 999) [NMEASInterface](#), NMEA system interface
 10018 (000 ÷ 999) [GLLSentence](#), Geographic Position sentence
- 10019 (000 ÷ 999) ModbusRTUSlave, modbus Rtu slave (ePLCUtyLib_B000, MNL151C020)
- 10020 (000 ÷ 999) [MWVSentence](#), Wind Speed and Angle sentence

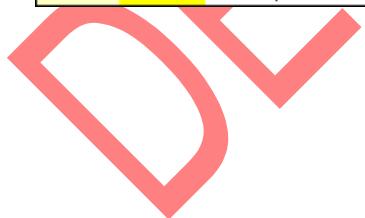
- 10030 (000 ÷ 999) [AuroraDSPMeasure](#), Aurora measure request to DSP
10031 (000 ÷ 999) [AuroraCEnergy](#), Aurora cumulated energy reading
10032 (000 ÷ 999) [sHWgSProtocol](#), HW group serial protocol
- 10033 (000 ÷ 999) ModbusAsciiSlave, modbus Ascii slave (ePLCUtyLib_B000, MNL151C020)
10034 (000 ÷ 999) [SysLogReport](#), send a report to Syslog server
10035 (000 ÷ 999) [StringToLogFile](#), store string to a log file
10036 (000 ÷ 999) [FileMemoryDump](#), dump memory on file
10037 (000 ÷ 999) [ModemPhoneCall_v1](#), executes a phone call
10038 (000 ÷ 999) [ModbusSlave](#), modbus slave
10039 (000 ÷ 999) [HIDClikDtaReader](#), HID RFID clock/data reader
10040 (000 ÷ 999) [MMasterDataTxfer](#), multimaster data transfer
10041 (000 ÷ 999) [DataTxferClient](#), Data transfer client
10042 (000 ÷ 999) [ModemHTTPGet](#), executes a HTTP Get request
10043 (000 ÷ 999) [SpyDataFile](#), spy data and stores them on a file
10044 (000 ÷ 999) [BroadcastDataSend](#), broadcast data send
10045 (000 ÷ 999) [StrainGaugeAcq](#), strain gauge acquisition
10046 (000 ÷ 999) [HMIBuiltInMessages](#), HMI built in messages
10047 (000 ÷ 999) [HMIBuiltInNetlog](#), Netlog HMI management
10048 (000 ÷ 999) [DataStreamExch](#), exchanges data between two I/O streams
10049 (000 ÷ 999) [HMIBuiltInTerminal](#), manages built in terminals
10050 (000 ÷ 999) [ccTalkProtocol](#), manages the ccTalk protocol
10051 (000 ÷ 999) [AlbericiAL66](#), Alberici AL66 coin acceptor
10052 (000 ÷ 999) [SNTPRequest](#), sends a SNTP request
10053 (000 ÷ 999) [ModbusTCPGateway](#), modbus TCP gateway
10054 (000 ÷ 999) [HTTPProtocol](#), HTTP protocol management

DEPRECATE

12.5 Ascii codes table

12.5.1 ASCII standard codes table

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	Ø	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	\$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(72	48	H	104	68	h
9	09	Horizontal tab	41	29)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	:	91	5B	[123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	
29	1D	Group separator	61	3D	=	93	5D]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□



12.5.2 ASCII extended codes table

Dec	Hex	Char									
128	80	ç	160	A0	á	192	C0	ł	224	E0	α
129	81	ü	161	A1	í	193	C1	ł	225	E1	ß
130	82	é	162	A2	ó	194	C2	τ	226	E2	Γ
131	83	â	163	A3	ú	195	C3	ㅏ	227	E3	ㅠ
132	84	ä	164	A4	ñ	196	C4	–	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	+	229	E5	σ
134	86	ã	166	A6	²	198	C6	Ւ	230	E6	μ
135	87	ç	167	A7	°	199	C7	Ւ	231	E7	ι
136	88	ë	168	A8	ڦ	200	C8	ڦ	232	E8	ڦ
137	89	ë	169	A9	ڻ	201	C9	ڦ	233	E9	ڦ
138	8A	è	170	AA	߱	202	CA	߱	234	EA	߱
139	8B	ି	171	AB	ି	203	CB	ି	235	EB	ସ
140	8C	ି	172	AC	ି	204	CC	ି	236	EC	ୟ
141	8D	ି	173	AD	ି	205	CD	=	237	ED	୧
142	8E	ା	174	AE	ୱ	206	CE	ଫ	238	EE	୯
143	8F	ା	175	AF	୳	207	CF	ଲ	239	EF	ଠ
144	90	ି	176	BO	ି	208	DO	ି	240	FO	ି
145	91	ା	177	B1	ି	209	D1	ି	241	F1	ି
146	92	କ	178	B2	କ	210	D2	ି	242	F2	ି
147	93	ଖ	179	B3	ଖ	211	D3	ି	243	F3	ି
148	94	ଙ	180	B4	ଙ	212	D4	ି	244	F4	ି
149	95	ଙ	181	B5	ଙ	213	D5	ି	245	F5	ି
150	96	ଔ	182	B6	ଔ	214	D6	ି	246	F6	ି
151	97	୹	183	B7	୹	215	D7	ି	247	F7	ି
152	98	ୟ	184	B8	ୟ	216	D8	ି	248	F8	ି
153	99	୪	185	B9	୪	217	D9	ି	249	F9	ି
154	9A	୨	186	BA	୨	218	DA	ି	250	FA	ି
155	9B	୮	187	BB	୮	219	DB	ି	251	FB	ି
156	9C	୰	188	BC	୰	220	DC	ି	252	FC	ି
157	9D	୩	189	BD	୩	221	DD	ି	253	FD	ି
158	9E	୮	190	BE	୮	222	DE	ି	254	FE	ି
159	9F	୤	191	BF	୤	223	DF	ି	255	FF	ି